

追補版

ver.1.10対応版

SX-WINDOW

プログラミング

吉沢正敏

SX-WINDOWプログラマのための福袋

SXer Tool Box付き

5" 2HD

SOFT
BANK







追補版

ver. 1.10対応版

SX-WINDOW

プログラミング

吉沢正敏

2X-WINDOW

●本書に掲載したプログラム名、システム名、CPU名などは一般に各社の登録商標です。
本文中では、とくにTM、Rマークは明記していません。

©1991 本書のプログラムを含むすべての内容は、著作権法上の保護を受けています。
著者、発行社の許諾を得ず、無断で転載、複製することは禁じられています。



はじめに

前著『SX-WINDOW プログラミング』が発売になって、ひと月もしないうちに X68000 の新シリーズ XVI がリリースされ、同時に SX-WINDOW も大幅に変更されてバージョン 1.10 となりました。新しい SX-WINDOW は、画面描画スピードの向上、プリンタマネージャ/プリンタドライバ周辺の充実、そして優秀なエディタの添付など、さらに実用性が高められた内容となっています。

バージョンアップ自体はおおいに歓迎すべきことなのですが、『SX-WINDOW プログラミング』だけではフォローしきれない部分もしだいに明らかになってきました。SX コールは大幅に増設され、新しいマネージャが2つも新設されています。

このような変更に対応するため、本書『追補版 SX-WINDOW プログラミング』が企画され、出版に至りました。さらに強力になった SX-WINDOW を、本書によって十二分に活用していただければ幸いです。

また、今回、サンプルプログラム類の入力の手間を省くことや、ペーパーメディアでは十分には伝わらない情報をお届けするため、付録としてフロッピーディスクを添付することになりました。このディスクには、パソコン通信で公開されている SX-WINDOW 用のフリーソフトウェアを、作者の皆さんの許可を得て収録させていただきました。

なお、本書では、『SX-WINDOW プログラミング』を『SX-WINDOW～』、『追補版 SX-WINDOW プログラミング』を『本書』あるいは『追補版』と呼んで区別することにします。

本書の内容

本書は、SHARP X68000 用にリリースされたウィンドウシステム、SX-WINDOW のアプリケーションを作成するために必要な情報をまとめたものです。とくに、SX-WINDOW のバージョンアップにともない、前著『SX-WINDOW プログラミング』ではカバーしきれなくなった部分について解説を行うことを目的としています。また、『SX-WINDOW プログラミング』で説明の不足していた、アプリケーション作成に關しての補足解説も行っています。

・第 0 章 SX-WINDOW ver1.10 の概要

バージョンアップされた SX-WINDOW の概要について述べています。内部的なことには深く立ち入らず、SX-WINDOW の歴史や外見的、機能的な変遷などについて述べ、どのように変化して現在に至ったかを示しています。

・第1章 プログラミングの補足説明

『SX-WINDOW プログラミング』で不足していたアプリケーション開発の手順についての補足解説を行っています。ここでは、アプリケーションの設計からデバッグまで、いくつかの段階に分けて解説を行います。

・第2章 拡張されたマネージャ

SX-WINDOW の機能アップにともない、既存のマネージャもそれぞれ大幅に機能の拡張が行われています。この章では、大幅に拡張されたグラフィックマネージャ、テキストマネージャ、タスクマネージャの3つを中心に、既存のマネージャの変更/拡張点について述べています。

新しい機能を利用したサンプルプログラムも掲載しました。

・第3章 新設されたマネージャ

SX-WINDOW バージョン 1.10 には、新しいマネージャとしてプリントマネージャ、サブウィンドウマネージャの2つが追加されています。この章では、それぞれのマネージャで導入された新しい概念や、それらを利用したプログラミングなどについて解説しています。

それぞれの新設マネージャを利用したサンプルプログラムも掲載しました。

・第4章 C 言語によるプログラミング

『SX-WINDOW プログラミング』に対してご要望の多かった、C 言語による SX アプリケーションの作成方法について述べています。処理系としては、XC バージョン 2.00 を利用しています。

・第5章 SX コールリファレンス

変更・追加された SX コールを1つ1つ解説します。

・APPENDIX

新しく追加されたリソースのリスト、いくつかのコードが追加されたリザルトコード表などを掲載しました。

付録ディスクについての解説もここで行っています。

付録ディスクの内容

付録ディスクは 2HD、1.2M バイトの Human フォーマットのディスク 1 枚で構成されており、次のような情報が収められています。

- ・C 言語による開発キット
- ・『SX-WINDOW プログラミング』、『追補版 SX-WINDOW プログラミング』のサンプルプログラムソース
- ・フリーソフトウェアの開発ツール
- ・フリーソフトウェアのアプリケーション

ご利用に際しては、APPENDIX の「付録ディスクの使い方」、およびディスク中の

README をよくお読みください。

使用環境

本書では、以下のシステムを独自に解析した結果をもとに解説を行っています。

- SX-SYSTEM Ver1.10 FSX.X 121548 91-03-15 12:00:00 *1
- SX-SHELL Ver1.10 SXWIN.X 16732 91-03-15 12:00:00 *1

筆者の使用した走行環境は、以下のとおりです。サンプルプログラム等は、以下の環境で正しく動作することを希望して作成されています。

- 本体 X68000 XVI-HD
- 増設 RAM CZ-6BE2A
- OS Human68k ver2.02
 - HUMAN.SYS 54240 90-05-05 12:00:00 *2
 - COMMAND.X 28026 90-05-05 12:00:00 *2

開発ツールとしては、以下のものを使用しました。サンプルプログラム等は、以下のツールを利用して正しく作成できることが確認されています。

- アセンブラ
 - AS.X 28194 88-04-02 12:00:00 *2
 - AS.X 99572 90-05-05 12:00:00 *2
 - HAS.X 29530 91-06-12 23:55:06 *3
- リンカ
 - LK.X 42598 90-05-05 12:00:00 *2
 - HLK.X 25782 91-09-24 21:21:24 *4
- デバッガ DB.X 38712 90-11-01 12:00:00 *2
- コンバータ CV.X 17570 90-05-05 12:00:00 *2
- リソースリンカ
 - RLK.X 12252 90-11-01 12:00:00 *1
 - RSC.X 16700 91-08-13 21:01:56 *5
- エディタ
 - MicroEmacsJ1.31
 - em.x 164108 90-04-24 11:00:00 *6
 - MicroEmacsJ1.40
 - em.x 244520 91-03-15 12:00:00 *7

フリーソフトウェアの作者の方々には、心から感謝いたします。

*1: (c) SHARP/First Class Technology

*2: (c) SHARP/Hudson

*3: フリーソフトウェア YuNK (中村祐一) 氏製作

*4: フリーソフトウェア SALT 氏製作

*5: フリーソフトウェア 清水和久氏製作

*6: フリーソフトウェア ICAM 氏, HOMY 氏製作

*7: フリーソフトウェア Iika/SALT/PEACE/SHUNA/rima 氏製作

CONTENTS

まえがき	3
------------	---

第0章	SX-WINDOW ver1.10 の概要	11
-----	------------------------------------	----

第1章	プログラミングの補足説明	17
-----	---------------------------	----

1...1	SX アプリケーションの発想	18
-------	-----------------------------	----

1	SX アプリケーションの発想	19
2	プログラムの基本仕様をまとめる	21
3	ユーザーインタフェースを決める	22
4	イベントへの対応を考える	23
5	必要な初期化処理, 終了処理をまとめる	34

1...2	コードの組み立て	38
-------	-----------------------	----

1	スケルトン	38
2	仕様からコードを起こす	47

1...3	モジュールの作成とリンク	75
-------	---------------------------	----

1...4	実行とデバッグ	89
-------	----------------------	----

1	ツールを利用しないデバッグ	92
2	DB.X バージョン 1.10 を利用するデバッグ	95
3	SXWDB.X を利用するデバッグ	99
4	ターミナルを利用するデバッグ	102

第2章

拡張されたマネージャ105

2...1 グラフィックマネージャ106

- 1 スクリーンタイプの追加106
- 2 ビットマップのバリエーションの追加106
- 3 図形の追加109
- 4 多様な画面モードへの対応111
- 5 フォントの拡張111
- 6 疑似ダイアログ114
- 7 下位ルーチンのユーザへの開放115

2...2 テキストマネージャ123

- 1 行と段落の概念123
- 2 テキストエディットレコードの変更123
- 3 段落情報127
- 4 編集履歴128
- 5 キャッシュ機能の追加128
- 6 アップデート処理の充実129
- 7 プロセステーブルの拡張130
- 8 そのほか135

2...3 タスクマネージャ137

- 1 モジュールヘッダの拡張137
- 2 起動時のレジスタ内容の変更138
- 3 タスク間通信の手順の変更139
- 4 タスク管理テーブルの拡張139
- 5 タスクマネージャイベントの拡張140
- 6 セルレコードに登録されるデータの種類の追加140
- 7 そのほか142

2...4 そのほかのマネージャ146

- 1 キーボードマネージャ146
- 2 リソースマネージャ146
- 3 ウィンドウマネージャ147

4	メニューマネージャ	147
5	コントロールマネージャ	148
2...5	サンプルプログラム	149
1	プログラムの仕様	149
2	プログラムの説明	151
3	プログラムリスト	152
第3章	新設されたマネージャ	169
3...1	プリントマネージャ	170
1	プリントマネージャの機能の概要	170
2	印刷の仕組み	175
3	プリントマネージャの利用	188
4	プリンタドライバの作成	198
5	まとめ	205
3...2	サブウィンドウマネージャ	206
1	サブウィンドウの意味	206
2	サブウィンドウの仕組み	208
3	サブウィンドウの利用	215
4	まとめ	220
3...3	サンプルプログラム	221
1	プリントマネージャのサンプルプログラム	221
2	サブウィンドウマネージャのサンプルプログラム	230

第4章 C 言語によるプログラミング245

4...1	C 言語とアセンブラの関係	246
--------------	----------------------------	------------

1	開発に必要な環境	247
4...2	C 言語による開発の制限事項	249
4...3	SXLIB.L	253
1	XC2 の出力する実行ファイルの構成	253
2	SXLIB.L のスタートアップ	254
3	SXLIB.L によってサポートされる関数	256
4	SXLIB.H	256
4...4	C 言語版スケルトン	257
4...5	サンプルプログラム	262
1	プログラムの仕様	262
2	プログラムの説明	262
3	プログラムリスト	264

第5章 SX コールリファレンス 273

SX コールリファレンスの利用法▶274

キーボードマネージャ▶275	リソースマネージャ▶275
ウィンドウマネージャ▶276	コントロールマネージャ▶277
メニューマネージャ▶278	サブウィンドウマネージャ▶279
プリントマネージャ▶280	テキストマネージャ▶285
タスクマネージャ▶297	グラフィックマネージャ▶305

APPENDIX 317

1	付録ディスク「SXer Tool Box」の使い方	318
2	SX1.10/EasyPaint で追加されたリソース	323
3	リザルトコード一覧	325
4	SX コール通巻索引	327

CONTENTS

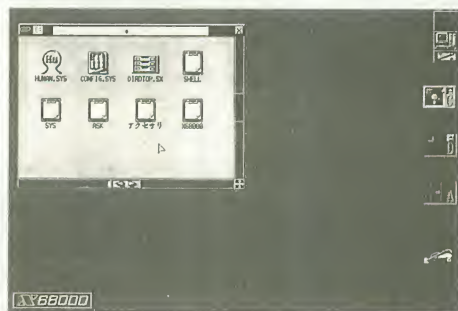
	あとがき	343
	欧文索引	344
	和文索引	344
COLUMN	SXWIN.X の未公開オプション	15
COLUMN	シェルの未公開操作	16
COLUMN	用語の変更	36
COLUMN	SX1.10 の不具合	37
COLUMN	前回の状態の再現の仕組み	70
COLUMN	make について	87
COLUMN	DB.X へのパッチ当て	104
COLUMN	TITLE.X の役割	121
COLUMN	HENWIN.X の役割と動作の仕組み	135
COLUMN	SRAM の内容	143
COLUMN	SYSDTOP.SX のフォーマット	144
COLUMN	GCC による開発	269
COLUMN	ライブラリのアセンブラからの使用	271

SX-WINDOW ver1.10 の概要

1991 年 4 月, XVI シリーズの登場とともに SX-WINDOW も大幅に改変され, バージョン 1.10 となりました。表示等のスピードアップやエディタ .X の添付などでいっそう実用的になったのはもちろんですが, 内部的にも大幅に改良/拡張されています。この章では, 現在までの SX-WINDOW の歩みと, 新しい SX-WINDOW, バージョン 1.10 の概要を示します。

X68000用ウィンドウシステムであるSX-WINDOWが最初にリリースされたのが1990年4月。ごく短い期間SUPER, EXPERTII, PROIIに添付されたバージョン1.00から始まって、すぐに1.02へバージョンアップ。これがいちおうの安定バージョンとして、単体で市販されるようになりました(図1)。

■図1 SX-WINDOW ver1.02のデスクトップ



マルチウィンドウ、マルチタスクのOSが、メモリわずか2Mバイト、しかもフロッピーベースで使えて、価格がわずか6800円(ただし限定価格)というのは、巷のウィンドウシステムの肥大化が始まりかけていた当時としても、驚異的ではありました*1。そんなわけで、致命的なバグもなく、入手も利用も容易なかたちで提供されたことから、SX-WINDOW ver.1.02(以下SX1.02)は比較的短期間で普及しました。

*1: もっとも、シングルタスクながら、Macも最初はメモリ128Kバイトが標準でし、Amigaは512Kバイトでマルチタスクまでも行っています。SX-WINDOWの場合は、システムプログラムをすべてRAM上に置かなければならないという点で多少不利ではあります。このような貧乏自慢のような比較はあまり意味がないかもしれませんが……。

SX1.02に問題があったとすれば、それは次のような点でした。

(1) 動作スピードが若干遅い

キャラクタベースのインタフェースと比較した場合、グラフィカルなユーザーインタフェースはどうしても処理が重くなりがちです。しかし、使う側の立場からすれば、それはつくる側の都合でしかありません。

(2) 基本的なデバイスの管理が不十分

OSの役割は資源の管理にあります。SX1.02には資源の一種であるデバイスの管理に不十分なところがありました。たとえば、基本的なデバイスであるプリンタの管理にもう1つ柔軟性がなく、アプリケーションが任意の図形を印刷しようとした場合は、SX-WINDOWの管理のおよばないIOCSレベルで処理を行う必要がありました。この方法ではタスク間の競合を避ける方法がないため*2、マルチタスク環境であるSX-WINDOWにはなじみません。

*2: そのタスクだけで CPU 時間を占有するという方法ではありますが、マルチタスクの環境下では、あまりスマートな方法とはいえません。

(3) グラフィックデータの表現力/扱いが弱い

マッキントッシュ以降のパーソナルコンピュータ上のウィンドウシステムは、多かれ少なかれ、マッキントッシュの影響下にあります。マッキントッシュの優れている点の 1 つに、文字データもグラフィックデータもほぼ同じように扱える、ということが挙げられます。

こうした優れた点はぜひとも取り入れてほしかったところですが、SX1.02 では、クリップボード（デスクトップスクラップ）でアイコンデータのほかには文字列データ以外はやりとすることが考慮されていませんでした。SX-WINDOW のようなグラフィカルな環境下で動作するワープロソフトが登場した場合、図形の張り込みがサポートされることは間違いありませんが、SX1.02 ではそこに機能的な穴が存在しました。

こうしたことから、SX1.02 は本格的なアプリケーションの動作する環境としては、やや力不足なものであったことは否めません。

しかし、それも 1991 年 4 月、X68000XVI シリーズとともに登場した SX-WINDOW ver.1.10（以下 SX1.10）によって過去の話となってしまいました（図 2）。

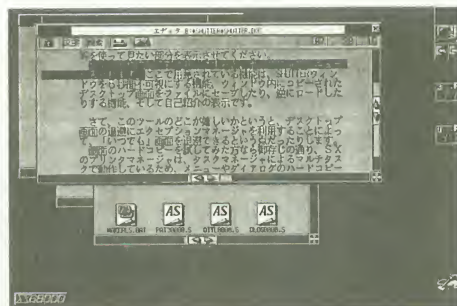
■図 2 SX-WINDOW ver.1.10 のデスクトップ



SX1.10 は、SX1.02 との上位互換性を維持したうえで、先ほど挙げた問題点をほとんどクリアしてしまいました。

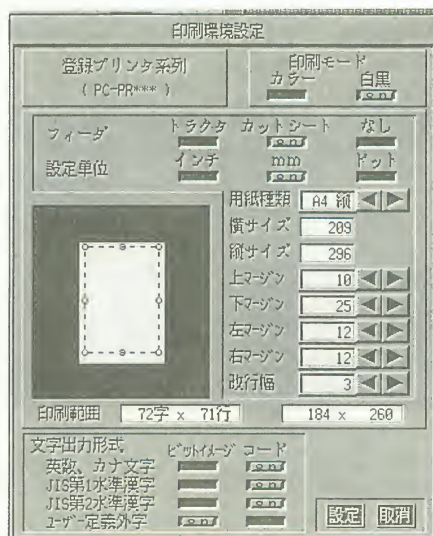
まず最初に気がつくことは、全体的な処理速度の向上です。その代表例が、SX1.10 から添付されるようになったエディタ X です（14 ページ図 3）。このエディタは Human 上のエディタと比較しても決してひけをとらないスピードで動いてくれます。SX-SYSTEM、とくにグラフィックマネージャとテキストマネージャが改良された結果、この速度の向上をもたらしています。

■図3 エディタ.X



そして、デバイスの管理については、プリンタに関しては大幅な改良が施され、細かいコントロールと表現力豊かな印刷が可能となりました。この機能は、SX-SYSTEM に新しく追加されたプリントマネージャによって提供され、すべてのアプリケーションがそれを利用することが可能です (図4)。

■図4 コントロールパネルの印刷環境設定



最後に挙げたグラフィックデータ云々については、SX-WINDOW 用アプリケーション (以下、SX アプリケーション) の第1弾である Easypaint-SX68K によって、それがもはや問題ではないことを見てとることができます (15 ページ図5)。Easypaint では複数のウィンドウ上でイメージを編集することが可能ですが、ウィンドウ間で任意の範囲のイメージをスクラップを経由してやりとりすることができます。

また、Easypaint の表現力はかなりのものがありますが、これは Easypaint の持つ力ではなく、改良されたグラフィックマネージャによる部分が大きいのです。

以上のことから、SX1.02 で問題と思われた部分の多くは SX1.10 によって解決、あるいは改善されていることがわかります。これはすなわち、SX-WINDOW の利用範囲がさらに広がった、ということの意味しています。

SX1.10 には多くの新しい機能が追加されています。それらについては第2章以降で解説することにしましょう。

■図5 Easypaint-SX68K



COLUMN | SXWIN.X の未公開オプション

SXWIN.X を起動する際には、コマンドライン上でいくつかのオプションを指定することができます。マニュアルには/E と/M が掲載されていますが、このほかに/K、/G などが有効です。

/K キーボードマネージャの OldOn の初期状態を 1 とする

/G<num> 画面モードを指定する

<num> は次のような値を指定する

bit5 4 ~ 0

□ □ □ □ □ □

	表示画面	表示色	同期周波数	実画面
0	512×512	16 色	31kHz	1024×1024
1	512×512	16 色	15kHz	1024×1024
2	256×256	16 色	31kHz	1024×1024
3	256×256	16 色	15kHz	1024×1024
4	512×512	16 色	31kHz	512×512
5	512×512	16 色	15kHz	512×512
6	256×256	16 色	31kHz	512×512
7	256×256	16 色	15kHz	512×512
8	512×512	256 色	31kHz	512×512
9	512×512	256 色	15kHz	512×512
10	256×256	256 色	31kHz	512×512
11	256×256	256 色	15kHz	512×512
12	512×512	65536 色	31kHz	512×512
13	512×512	65536 色	15kHz	512×512
14	256×256	65536 色	31kHz	512×512
15	256×256	65536 色	15kHz	512×512

16	768×512	16色	31kHz	1024×1024
17	1024×424	16色	24kHz	1024×1024
18	1024×848	16色	24kHz	1024×1024
0	表示画面モード			
1	実画面モード			

91 ページで述べるように、SXWIN.X と SXWDB.X は同じものです。外部カーネルとして SXWIN.X (SXWDB.X) を起動する場合には、以上のほかに、次のオプションを指定することができます。

- /D SXWIN.X (SXWDB.X) をデバッグモードで起動。シェルを起動したプログラムをタスクとして再起動する。SXWDB.X として利用する場合には必ず指定すること
- /R リソースのオープン指定
 - /R1 SYSTEM.LB をオープンする
 - /R3 SYSTEM.LB, BUILTIN.LB をオープンする
 - /R5 SYSTEM.LB, ICON.LB をオープンする
 - /R7 SYSTEM.LB, BUILTIN.LB, ICON.LB をオープンする
- /L リソースのメモリへのロードの指定
 - /L0 リソースをメモリにロードしない
 - /L1 SYSTEM.LB をメモリにロード
 - /L2 BUILTIN.LB をメモリにロード
 - /L3 SYSTEM.LB, BUILTIN.LB をメモリにロード
 - /L4 ICON.LB をメモリにロード
 - /L5 SYSTEM.LB, ICON.LB をメモリにロード
 - /L6 BUILTIN.LB, ICON.LB をメモリにロード
 - /L7 SYSTEM.LB, BUILTIN.LB, ICON.LB をメモリにロード

COLUMN シェルの未公開操作

すでにご存じの方も多いかと思いますが、SX1.10 となって、システムアイコン、デスクトップアイコン、ページアイコンをドラッグすることが可能になりました。

[OPT.1] キーを押しながら、それぞれのアイコンを左ボタンでドラッグして、お好みの位置に配置してください。

第 1 章

プログラミングの補足説明

SX-WINDOW のアプリケーションの開発については、前著『SX-WINDOW プログラミング』でもひとつとおり解説しましたが、ここではさらに実践的な解説を行い、『SX-WINDOW プログラミング』での解説を補足することになります。

SX-WINDOWの具体的なプログラム作りに入る前に、わたしたち人間の発想をどのようにしてSX-WINDOWのアプリケーションに反映させて行くのかを考えてみましょう。SX-WINDOWのプログラム作りだけでなく、アプリケーションをつくる人にとって大切でしかも切実な問題のはずです。

最初は、一般的な話から始めましょう。

プログラムをつくるという作業には、人間の頭の中身（のごく一部）をコンピュータ上に移植し、目に見えるようにするという側面があります。熟練した職人さんの作業の段取りやテクニックをプログラム化してロボットに移植してやることにより、ロボットは職人さんと同じように働くことができるようになります。また、物理学者の理論をプログラム化してシミュレーションを繰り返すことによって、その正しさを検証することができます。

ところが、現在のノイマン型のコンピュータは、人間の頭脳とはまったく違う方法で計算やデータの加工を行っているため、人間の頭の中をプログラム化するといっても、そうかんたんなことではありません。たとえば、「 2×3 」という計算であれば、人間の場合はよくわからない複雑な過程を経て、しかも、それを意識することもなく、かんたんに「6」という答えを出すことができます*1。これをコンピュータで行うとすると、賢明な読者の皆さんであれば、かんたんに

```
move.w    # 2,d0
mulu      # 3,d0
```

のようなプログラムを導き出すことができると思います。しかし、この計算の方法は、人間が頭の中で行っている方法とはまったく別物であろうことはほぼ間違いありません。このようなプログラムを導き出すことができたのは、皆さんが頭の中で行っている計算の作業を逐一追っていた結果ではなく、「コンピュータで計算を行う場合、レジスタに値を入れて計算命令を実行すればよい」という知識があったからにはかなりません。つまり、計算という「概念」から「コンピュータでのやりかた」という知識への橋渡し、という変換プロセスを経た結果であるといえるでしょう。

*1:日本人の場合、「にさんがろく」というテーブルを参照しているケースが多いのだらうという予想までは可能ですが、このテーブルにアクセスするまでの過程やその方法が明らかになるためには専門家の先生たちの研究を待つほかありません。

いまの例のアセンブラのような、コンピュータの世界の中でも、いわゆる低級なところでの「やりかた」に「概念」を変換していくのは、コトバの使い方がかけ離れているという意味で、多かれ少なかれ、人間にとって苦痛です。このコトバの距離をいくらかでも縮めるために、高

級言語と呼ばれるものが登場してくるわけですが、それでも事の本質は変わりません。人間の頭の中にある、ぐにやぐにやした概念を、きちんとしたコンピュータの世界の鋳型にはめるために、コトバはコミュニケーションの手段とはなりますが、その内容こそが問題だからです。

プログラムが上手に組めるかどうかは、この「概念」をいかにうまく分析できるか、そして、それをいかに上手に「コンピュータでのやりかた」に変換するかにかかっています*2。後者は、いってしまえば、機械的な経験の蓄積の問題であり、多くのプログラムに触れることで、“ある程度”のレベルには誰でも到達できるはずです。問題は、前者です。後者の経験を活かすためには、「概念」を分析し、深く考える習慣が必要となってくるでしょう。逆に、はじめに「概念」をよく分析しておきさえすれば、それ以降の作業がかなり楽になるということもできます。おそらく、今後、プログラム開発の自動化が進むにしたがって、人間の仕事は前者に集約されていくことでしょう。決まりきったような処理であれば、すべてコンピュータが自分でプログラムを生成してしまうことも十分考えられます。人間がプログラムを組む意味は、まさにこの概念を分析するところにこそあるのです。

*2:組織的にソフトウェアを開発する場合、分業体制で作業を行うわけですが、前者をシステムエンジニアが、後者をプログラマーが担当していると考えられます。

さて、“概念の分析”などという固い言葉を使ってしまったましたが、そのポイントを平たくいえば、「どんなプログラムをつくりたいのかをはっきりさせる」ことに尽きると思います。SX-WINDOW のアプリケーション (以下、SX アプリケーションと呼びます) もコンピュータのプログラムに違いはありませんから、その開発はここからスタートします。

とりあえず、これから始める解説のための例となるプログラムを決めておきましょう。SX アプリケーションとしての一般性を持ち、かつ、掲載できる程度に小規模で、実用的でもあるものとして、いわゆる「時計」プログラムをつくることにします。標準で SX-WINDOW に付属する「時計.x」もあることですから、多少ひねったものにしましょう。

この段階では、かならずしもコンピュータに向かう必要はありません。下手にコンピュータが使える状態でそばにあれば、よく考えないうちにコードを書き始めてしまう場合もありますから、むしろコンピュータは電源を切って、紙とエンピツだけで作業を行うほうがよいかもしれません*3。

*3:自戒を込めつつ…。

1 SXアプリケーションの発想

SX アプリケーションは、従来の Human 用のプログラムとはずいぶん様子が異なります。Human 用のプログラムの場合、ユーザからの 1 行文字列を入力してもらう必要ができたとき、C であれば、

```
gets(strings);
```

と書けばすみません。しかし、SX アプリケーションではこうはいきません。なぜ、SX でのやりかたが異なるのか、どうすればよいのかを考える前に、逆に、なぜ Human 用のプログラムはこれでいいのかを考えてみることにしましょう。

Human は従来のキャラクタベースのインタフェースを想定した OS をモデルにつくられています。もともと、そういったシステムというのは、tty 装置というタイプライタの化け物のような装置に端を発しており、人間とのコミュニケーションといえば、コンピュータ側が 1 行たずねる（タイプライタの化け物が 1 行打ち出す）と、人間が 1 行答える（キーボードをバチバチ打って改行キーを押す）といった程度のものでした。Human の DOS コールを眺めていると、その痕跡を認めることができると思います。

このような状況下では、人間からの入力进行待つということは、ごく素朴に、リターンキーが押されるまで入力された文字をバッファリングする、という悠長なものでした。シングルタスクの OS というのもあって、ほかのプログラムが CPU 時間を求めてせかしているわけではないし、プログラムは悠々と自分の仕事に専念することが許されていたのです。

しかし、時は流れて、世は GUI の時代になりました。GUI というのは、要するに、使う人間の自由度が増す、くだけた言い方をすれば、人間のわがままをコンピュータはすべて聞いてやらなければいけない、ということにほかなりません。人間は、ロール紙の端で、コンピュータが聞いてくる質問に答えるだけの生活には飽き飽きしています。もっと積極的に、画面上にいくつも表示されている操作パネルのあっちを（マウスで）押し、こっちに（キーボードで）書き込み、自由に仕事をしたいと考えるようになったわけです。

こうなると、コンピュータも悠長にリターンキーを待っているわけにはいきません。人間の動きにつねに気を配って、ここを（マウスで）押されたらこういう仕事をして、キーボードから文字が入力されたら適当な場所の文字列に追加して……というぐあいに、めまぐるしく働かなくてはいけなくなりました。

先ほどの `gets()` のような方法は、GUI には不向きです。`gets()` で人間が 1 行入力し終わるまで、ほかの仕事が何もできないからです。移り気な人間は、入力している途中で、画面上のどこかのボタンを押したくなるかもしれません。GUI を標榜するプログラムならば、それを許さなければならないのですが、`gets()` ではどうにもなりません。

そこで、人間の気まぐれな行動に逐一对応するために採用されたのが、SX-WINDOW の場合、イベントというわけです。人間がマウスのボタンを押したり、キーボードのキーを押したりするたびに、プログラムにはイベントというかたちで通知されます。プログラムはそのたびに、それに応じた行動をとれば、人間のわがままに追従することができます。

すなわち、`gets()` のかわりに、

イベント発生まで待つ。

- レフトダウンイベントだったら…
- ライトダウンイベントだったら…

- キーダウンイベントだったら、そのキーコードを文字列に追加する。リターンキーだったら、文字列の入力処理を終了して次のステップに進む。

という処理の流れが考えられます。

SX アプリケーションをつくる場合は、人間がある行動をしたときにはどのように対応したらよいか、という視点でものを考えることが必要であることを肝に命じておいてください。

2 プログラムの基本仕様をまとめる

開発が始まったいまの時点で決まっていることといえば、

- ・ SX-WINDOW 上で動作する時計
- ・ “時計.x” よりは多少ひねる

という 2 点だけです。これではあまりに曖昧ですから、もう少し中身を詰めてみましょう。

- ・ デジタル時計とする

アナログ時計のほうがプログラムとしてはおもしろくなりますが、針を描画したりする手法を解説するのが目的ではないことと、コードが大きくなってしまふことから、デジタル時計とすることにしました。

「多少ひねる」という点について、もう少し明確にして、

- ・ 12 時/24 時制を切り替え可能とする
- ・ 毎正時に時報を鳴らす。時報の on/off も切り替え可能とする

というところではいかがでしょうか。

また、SX-WINDOW の特徴の 1 つである、「SX-WINDOW 終了直前の環境が次に起動したときに再現される」という点もきちんとサポートすることにしましょう。具体的には、

- ・ 12 時/24 時制の設定や、時報の on/off の設定などを、SX-WINDOW を終了しても記憶しておくようにする

ということです。

基本的な仕様としては、この程度決めておけば十分でしょう。

3

ユーザーインターフェースを決める

SX アプリケーションは、ユーザにとって使いやすいものでなければなりません。前著『SX-WINDOW プログラミング』(以下、『SX-WINDOW〜』とします)でも述べたように、そのためにアプリケーション作成者が意識しなければならない約束事(ガイドライン)が定められています。ユーザーインターフェースを設計する際には、『SX-WINDOW〜』の230ページ以降の記述を参照し、これに準ずるようにしてください。

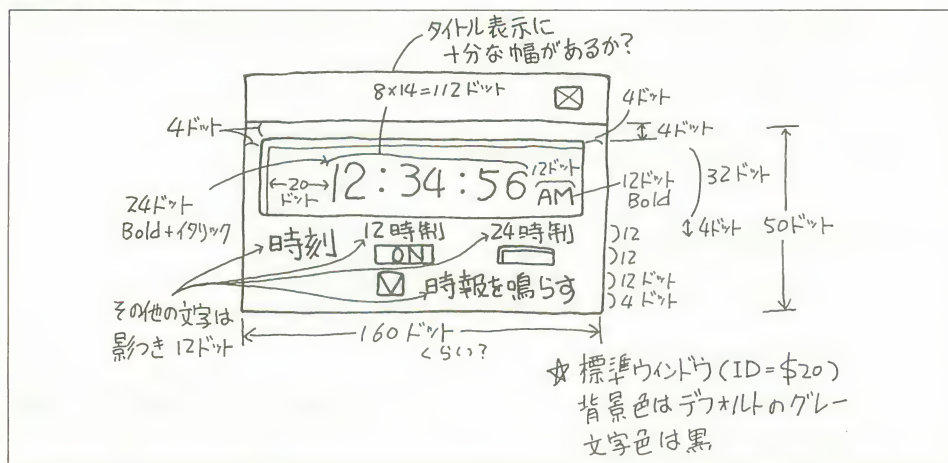
以上のことを頭に置いて、私たちの「時計」のユーザーインターフェースを決めることにします。

なんらかの情報を表示する場合、SX-WINDOW ではウィンドウを開いて、そこに描画を行うことが原則となっています。私たちの「時計」も例外ではありません。

ウィンドウ内部に表示する情報としては、時刻、そして AM/PM 表示が挙げられます。これだけでしたら、ウィンドウ ID\$10 の時計用ウィンドウでもよいのですが、12/24 時制、時報 on/off を設定するためのコントロールを置くことを考えて、ウィンドウ ID\$20 の標準ウィンドウを使うことにします。

こうして紙の上にウィンドウ内部の情報等の配置を描いてみたのが図1です。

■ 図1 画面設計の例



ウィンドウ下部には、12/24 時制を設定するためのラジオボタン*4、時報 on/off を設定するためのチェックボックス*5が並んでいます。これらは、設定する情報の性質にしたがってコントロールの種類を決めています。

*4:『SX-WINDOW〜』では、「セレクトボタン」と呼んでいましたが、用語が変更になりました(36ページのコラム参照)。

*5:『SX-WINDOW〜』では、「オルタネイトボタン」と呼んでいましたが、用語が変更になりました(36ページのコラム参照)。

ラジオボタンは、「12 時制」、「24 時制」というタイトルのついた 2 つを 1 グループとし、どちらか一方を選択します。チェックボックスは、「時報」というタイトルのついたものの on/off を選択することで、時報を鳴らす/鳴らさないを決定します。

文字やボタンを配置する座標を正確に決めておく必要はありません。だいたいの座標を決めておいて、後で調整するとよいでしょう。

4 イベントへの対応を考える

『SX-WINDOW〜』でも述べたように、SX アプリケーションは「イベント駆動方式」によって動作します。アプリケーションが対応すべきイベントにはどのようなものがあったか、思い出してみてください。

・アイドルイベント	→	ほかのイベントが発生していないことを示す
・レフトダウンイベント	→	マウスの左ボタンが押されたときに発生
・レフトアップイベント	→	マウスの左ボタンが離されたときに発生
・ライトダウンイベント	→	マウスの右ボタンが押されたときに発生
・ライトアップイベント	→	マウスの右ボタンが離されたときに発生
・キーダウンイベント	→	キーボードが押されたときに発生
・キーアップイベント	→	キーボードが離されたときに発生
・アップデートイベント	→	アップデートが発生したときに発生
・アクティブイベント	→	ウィンドウがアクティブになったときに発生
・システムイベント 1	}	タスクマネージャや、ほかのタスクから
・システムイベント 2		メッセージが送られてきたときに発生

以上の 11 個がありました。

これらのそれぞれについて、私たちの時計がどのように対応すべきか、考えてみましょう。やはり、紙などを用意して、「このイベントでは、このような順番で、こういう処理をする」的なことを書き出しておくといでしょう。

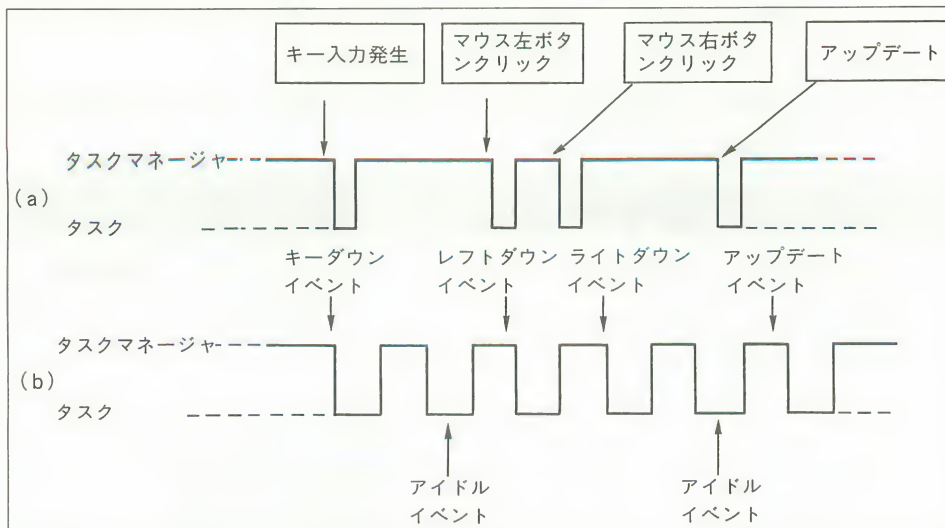
① アイドルイベント

アイドルイベントは、ほかのイベントが発生していない場合に、定期的に発生するイベントです。

アイドルイベント以外のイベントは、そのイベントが発生させる出来事が起こったときにはじめて発生します。この出来事は続けざまに発生するわけではないので、これらのイベントは、ある程度の一定でない間隔をおいて散発します (24 ページ図 2 (a))。アイドルイベントは、その間隔を埋めるようにして発生します (24 ページ図 2 (b))。

SX アプリケーションは、何かイベントが発生したら、短い時間だけ、それに対応する仕事

■図2 アイドルイベント発生の様子



を行い、その仕事が終わったら、すみやかにイベント待ちに戻ることがガイドラインによって求められています。すべての SX アプリケーションが、そのルールを守ることによって、マルチタスクがスムーズに行われることになります。このような仕事のしかたを決められている SX アプリケーションにとって、アイドルイベントは特別な意味を持っています。

アイドルイベント以外のイベント、たとえば、レフトダウンイベントによってだけ動作するアプリケーションを考えてみましょう。

レフトダウンイベントは、マウスの左ボタンが押されないかぎり発生しません。したがって、このアプリケーションはマウスの左ボタンが押されたときにだけ少し仕事をして、そのわずかな時間以外は延々とイベント待ちを繰り返すことになります。イベント待ちの間、アプリケーションは停止しているように見えますから、表面的にはなににも仕事もしていないように見えます。

このような動作でもかまわないアプリケーションはたくさんあります。たとえば、SX-WINDOW に標準で付属するユーティリティなどは大部分がこの類いです。“タイプ.X” はスクロールバーを操作しないかぎりじっとしていますし*6，“コントロールパネル.X” もボタンを左クリックしないかぎり何もしていないように見えます。つまり、これらはユーザから働きかけないかぎり、動作していないのと同じことになります。しかし、このような動作では困るタイプのアプリケーションもこの世には存在します。

*6: SX1.10 からは表示されている文書をクリック、ドラッグして内容をカット & ペーストできるようになっていますから、この表現は正確ではありません。

たとえば、“暁子.X” です。起動してみて、しばらく手を触れずに画面を眺めてみてください。手を触れていない、つまり、イベントを発生させる出来事を起こしていないにもかかわらず、ウィンドウの中では暁子さんは自転車で走り続けているはず。このような、ある程度

リアルタイムに、「勝手に動く」ような動作を行わなければならない SX アプリケーションは、アイドルイベントを受け取ったときに、その動作を部分的に、少しずつこなしていくのが定石です。

“暁子”の場合ならば、アイドルイベントのたびに、プログラムは次のような処理を行っているはずで

- 1) イベントレコードからイベントの発生した時刻を得る
- 2) 前回暁子さんの絵を書き換えたときに保存しておいた時刻と比較して、ある一定時間以上経過しているかどうかを調べる
経過していなければ、そのままイベント待ちに戻る
- 3) ある一定時間が経過していた場合、前の暁子さんの絵を消して、次の位置に描画する*7
- 4) 今回のイベント発生時刻を保存してイベント待ちに戻る

*7:暁子さんが自転車漕いでいるように見せるために、前回とは異なった絵を描画しているはずですが、実際には暁子さんだけでなく、犬の移動処理も行っているはずですが、ここでは割愛します。

このような処理をアイドルイベントが発生するたびに行うことによって、暁子さんは一定の時間間隔で、少しずつ移動するであろうことはわかりいただけると思います。

私たちの時計にも、このようなリアルタイム的な、「勝手に動作」する部分があります。いうまでもないことですが、時刻の表示は少なくとも 1 秒ごとに書き換えなくてはならないからです。先ほど暁子さんのアイドルイベントの処理を挙げたように、私たちの時計のアイドルイベント発生時の処理を考えてみましょう。流れとしては、ほとんど “暁子.X” と同じです。

- 1) イベントレコードからイベントの発生した時刻を得る
イベントレコードに収められている時刻は、SX-SYSTEM が立ち上がったからの経過時間を 1/100 秒単位で示したものですから、時刻表示の書き換えタイミングを計る目安にはなりますが、そのまま時刻として表示に流用できるわけではありません。
- 2) 前回時刻表示を書き換えたときに保存しておいた時刻と比較して、1 秒以上経過しているかどうかを調べる
経過していなければ、そのままイベント待ちに戻る。

イベントレコードに収められていた時刻と、前回保存しておいた時刻を比較して、1 秒以上経過しているかどうかは

$$(\text{今回の時刻} - \text{前回の時刻}) \geq 100_{(10)}$$

という条件式を評価することで確かめることができます。この式を評価した結果が真の場合、前回の書き換え以来、 $100 \times 1/100 \text{ 秒} = 1 \text{ 秒}$ 以上経過していることになり、次の処理である時刻表示の書き換えを行います。逆に結果が偽である場合は、まだ書き換えを行う必要がないので、そのままイベント待ちに戻ります。

3) 前の時刻表示を消して、現在時刻を描画する

SX コールには現在時刻を得るという機能は用意されていないので、IOCS コールを利用することになります。IOCS コールの\$56 番で得られる時刻は特殊な数値で返ってくるので、IOCS コールの\$57, \$5B 番を呼び出して、これを文字列に変換します。こうして得られた文字列を、SX コールを利用してウィンドウ内に描画するわけですが、その前に前回描画した時刻を背景色で塗り潰し、消しておくことを忘れてはいけません*8。キャラクタ型のディスプレイ装置を使い慣れてきた方は注意が必要かもしれません。

この処理はグラフィックマネージャを利用するだけなので、ここで細かく説明するまでもないでしょう。

*8: SX-WINDOW では、文字列を表示し、その後、同じ場所に文字列を上書きすると、文字が「重ね打ち」状態になってしまいますが、Human の場合は、前に表示されていた文字列は完全に消えてしまい、後から書いた文字列だけが残ります。

もっとも、Human にかぎらず、キャラクタ型の（またはそれをエミュレートした）ディスプレイ装置では、一般に、このように「重ね打ち」状態にはなりません。

4) 今回のイベント発生時刻を保存してイベント待ちに戻る

このイベントが発生したときに、イベントレコードの中に納められていたシステム時刻をワークの中に保存します。この値は、次にアイドルイベントが発生した際の 2) のステップにおいて参照されることになります。

以上のような処理で、見かけ上、どのような動作が実現できるか、あらためて説明することもないでしょう。

さて、時刻の書き換えのほかに、私たちの時計にはもう 1 つ、リアルタイムに行わなければならない仕事がありました。それは、時報を鳴らす時刻のチェックです。時刻は刻々と変化するので、少なくとも 1 秒ごとには時報を鳴らすかどうかを調べなければなりません。ということで、時刻が変化するたび、すなわち時刻の表示を書き換えるたびに、それが正時 (xx 時 00 分 00 秒) であるかどうかを調べ、そうであった場合は時報を鳴らす、という処理を行うことになります。私たちの時計では、時報を鳴らすか鳴らさないかを設定できるようになっているので、その設定を調べて、時報を鳴らす処理を行うか行わないかも決めなければなりません。

先ほどの処理の流れの中の 3) と 4) の間で、次のように処理を行います。

3.5) 時報を鳴らすかどうか調べる

鳴らさなくてよい場合は、4) へ。

鳴らす場合は、時刻の分、秒の桁を調べて、両方とも 0 の場合、BEEP 音を鳴らす。

以上でアイドルイベントで行うべき処理が決まりました。必要と思われるワークエリアの見当もついてきたので、それも紙に記録しておきます。

②レフトダウンイベント

レフトダウンイベントは、マウスの左ボタンが押し込まれたときに発生するイベントです。私たちがマウスのボタンを操作するとき、「押して離す」を 1 アクションとしてとらえていますが、SX-SYSTEM にとっては、「押す」と「離す」ことの、2 アクションとして認識されています。レフトダウンイベントは、このうちの「押す」が起こった場合に発生するイベントであることを明確に理解しておいてください。

時計として動作するための必要最低限の処理のうち、大部分はアイドルイベントで行うことはおわかりいただけたと思います。しかし、SX-WINDOW という OS 上で動作するアプリケーションである以上、ほかにもしなければならない仕事があります。

たとえば、SX アプリケーションは原則としてウィンドウ内部で動作します。多くの場合、標準ウィンドウを利用することになりますが、ユーザがウィンドウ自体をマウスで操作した場合、標準ウィンドウは次のように動作することが求められています。

- ・ウィンドウ上で左クリックされたらアクティブになる
- ・ドラッグリージョン上で左ボタンを押されたら、ウィンドウをドラッグ
- ・クローズボタンを左ボタンで押されたら、ウィンドウをクローズ
- ・サイズボタンを表示しているウィンドウは、左ボタンによるダブルクリック、ドラッグに応じて、それぞれズームイン/アウト、リサイズを行う

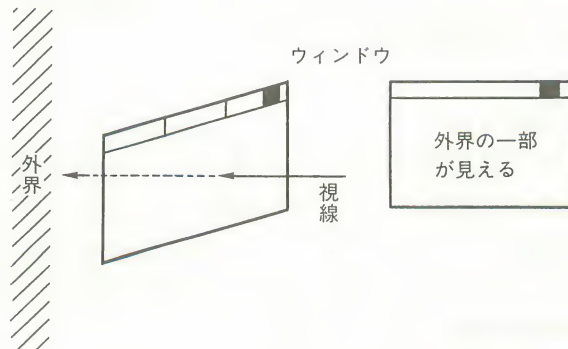
以上の 4 つを見ればわかるように、ウィンドウ自体を操作するときには、すべてマウスの左ボタンを用いることになっています。つまり、マウスの左ボタンが押された場合、その場所によっては、アプリケーションは純粋な自分の仕事以外の仕事、いってみれば「雑役」的な仕事をしなければならないのです。つまり、ウィンドウは（少なくとも、ウィンドウの枠と付属物は）OS という家の壁に張り付いた、OS の世界の一部分ですから、そこを操作された場合は、OS のための仕事をしなければなりません（28 ページ図 3）。

一方、窓であるところのウィンドウのガラス部分（ウィンドウコンテンツ）から見えているのは、アプリケーションの世界の風景です*9。ここに見えているボタン等のコントロール類も、やはり左ボタンで操作することになっています。私たちの時計も、ラジオボタン 2 つと

チェックボックス 1 つの計 3 つを持っています。これらを左ボタンで操作された場合 (= レフトダウンイベントが発生した場合)、アプリケーションはこれに対応した処理を行う必要があります。

*9: このあたりのたとえについては、「SX-WINDOW～」の 89 ページを読み返してみてください。

■図3 ウィンドウの概念



つまり、レフトダウンイベントが発生したとき、そのときのマウスポインタの位置にしたがって、アプリケーションはいくつかの処理に分岐しなければならないことになります。

ところで、レフトダウンイベントは、かならずしも自分のウィンドウに関係がある場合ばかりではありません。まったく関係のない、ほかのウィンドウ上でボタンが押された場合でもイベントは通知されます。そのため、実際には次のような流れで処理の分岐を行うことになります。

- 1) イベントが発生したとき、マウスポインタは自分のウィンドウ上にあったかどうかを調べる。自分のウィンドウ上になれば関係ないので、イベント待ちに戻る
- 2) ウィンドウはそれまでインアクティブだったかどうかを調べ、インアクティブであったのなら、アクティベートする
- 3) マウスポインタの位置をさらに細かく調べ、ウィンドウ上のどこであったかを調べる。ウィンドウコンテンツ内であれば、アプリケーション独自の処理を行い、ウィンドウの枠の上であれば、定型的なウィンドウ操作の処理に分岐する
- 4) それぞれの処理が終わったら、イベント待ちに戻る

どのようなアプリケーションも、おおむね、このような流れでレフトダウンイベントに対応していると考えられます。

いろいろと調べものが多いように思えますが、このあたりは案外かんたんに調べることができるので、案ずるにはおよびません。具体的なコーディングについてはしかるべきときに述べるとして、ここでは私たちの時計のために、3) のあたりをもう少し細かく決めてしまいま

しょう。

まず、マウスポインタがウィンドウコンテンツ内にある場合について。

私たちの時計は、先ほども述べたように、3つのコントロールを持っています。ウィンドウコンテンツ内でのマウスポインタの位置としては、次の3つが考えられます。

- a) ラジオボタン 1 の上
- b) ラジオボタン 2 の上
- c) チェックボックスの上
- d) それ以外

d) の「それ以外」であった場合、とくにすることは決めていないので、そのままイベント待ちに戻ってしまえばよいでしょう。

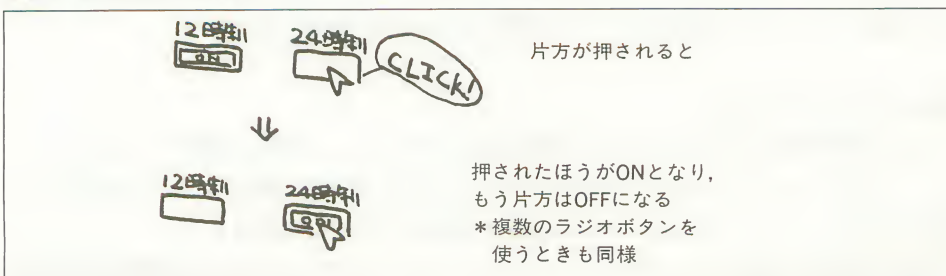
a)~c) のようにコントロール上であることが確認された場合は、共通してしなければならぬ処理があります。コントロールの多くは、それが「操作された」と判断するためには「押された」だけでは不十分で、「離された」ところまで確認する必要があります。たとえば、標準ボタンならば、標準ボタンの上でマウスの右ボタンが押され、かつ同じ標準ボタンの上で離された場合に、はじめて「操作された」ことになります。同様に、私たちの時計で利用するラジオボタン、チェックボックスも「操作された」ことを確認しなければなりません^{*10}。

^{*10}: このほかのコントロールの場合もほぼ同様です。スクロールバーやスライドボリュームなど、ドラッグできる部分（サム）を持っているコントロールの場合、左ボタンが押されて、離されたときにはドラッグがすすんでいることも考えられます。

一見、面倒に思えますが、操作されたことを確認する SX コールが用意されているので、それと呼び出すだけの話です。この結果、結局、「操作された」ことが確認できなかった場合、ここまでの一連の操作には意味がなかったことになり、そのままイベント待ちに戻ります。

a), b) の場合、ラジオボタンのガイドラインにしたがって、今回操作されたボタンを on にして、もう片方は off にするという処理を行う必要があります (図 4)。そして、どちらが on になったかを、ワークエリアに記録します。前回と異なる時制を意味するラジオボタンが on になった場合、表示している時刻を書き換えなければなりません。アイドルイベントでも

■図 4 ラジオボタンの操作

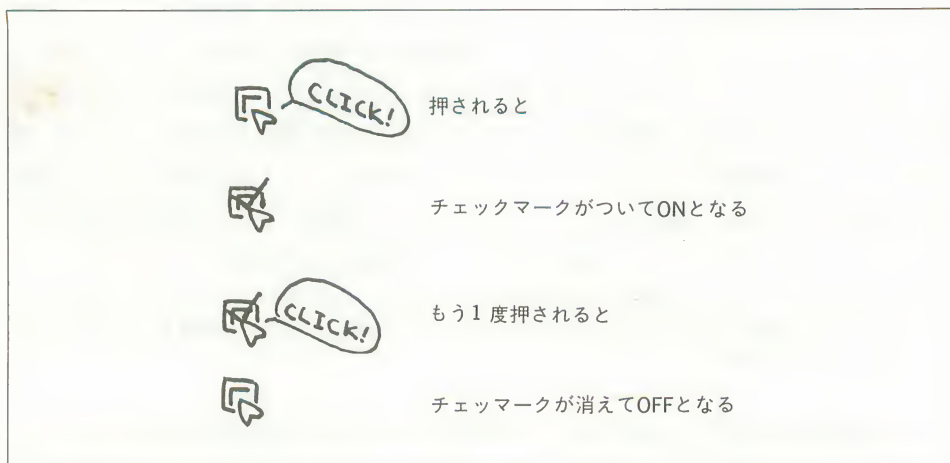


時刻の書き換えを行いますから、時刻描画ルーチンをサブルーチンとして用意しておくとしきります。

このように、複数箇所で行いたいようなことをする場合、サブルーチンとしてどんどん分割することは、SX 上にかぎらず、プログラミングの常道ではあります。もっとも、やりすぎると、かえってプログラムは読みにくくなりますから、ほどほどにしておいてください。

c) の場合、チェックボックスのガイドラインにしたがって、操作されるごとにチェックマークの on/off を切り替える処理を行います (図 5)。この結果は、時報を鳴らすか鳴らさないかを示すワークエリアに保存します。アイドルイベント発生時には、ここを参照することになります。

■図 5 チェックボックスの操作



ウィンドウの枠 (と付属物) の上にマウスポインタがあった場合は、ほとんど定型処理であり、SX コールの中で処理されてしまうことが多いので、あまり細かくは考えなくてもよいのですが、先ほど挙げたいくつかの動作を行わなければならない、ということだけは忘れないようにしてください。紙の上には「ウィンドウに関する処理」とでも書いておけば、それでいいかもしれません。

レフトダウンイベントは、どうしてもそれに対応して行うべき処理が多くなる傾向があります。混乱しないように、場合分けをして整理することをおすすめします。

③レフトアップイベント

SX-SYSTEM がマウスのボタンの操作を 2 アクション、「押す」+「離す」でとらえていることはすでに述べましたが、レフトアップイベントは、後者の「離す」によって発生するイベントです。レフトアップですから、左ボタンが離されたときに発生します。

私たちの時計の場合は、このイベントではとくに何もしません。左ボタンが離されたときに

何をするか、とくに決めていませんでした。「コントロールが押され、離されるのを確認して『操作された』ことを確認するはずではなかったの?」と思われるかもしれませんが、コントロールが「操作された」ことを確認する仕事は SX-SYSTEM が行ってくれるので、アプリケーションはとくにこのイベントを利用する必要はないのです。

レフトアップイベントを利用するのは、何かをドラッグした場合、どこで離されるかを知る必要がある場合などです。

④ ライトダウンイベント

⑤ ライトアップイベント

マウスの右ボタンについて、レフトアップ/ダウンイベントと同様な意味を持つイベントです。

マウスの右ボタンは、アクティブなウィンドウに付随するポップアップメニューを操作するために使われますが、私たちの時計ではポップアップメニューを使いませんから、これらのイベントではとくに何も行いません。

⑥ キーダウンイベント

⑦ キーアップイベント

キーボードのキーが押しこまれた場合に発生するのがキーダウン、離されたときに発生するのがキーアップイベントです*11。

文字列の入力や、メニューのショートカットなどを行うアプリケーションであれば、これらのイベントに対応する必要がありますが、私たちの時計には関係がありません。

*11:ただし、筆者はキーアップイベントが発生するところを確認していません。

⑧ アップデートイベント

ウィンドウを持っているアプリケーションは、かならずアップデートイベントをサポートしなければなりません。アップデートイベントは、デスクトップ中に存在するウィンドウの中のどれかにアップデートを行う必要ができた場合に発生します。アップデートについて、ここで解説をすることはしません。アップデートは重要な概念ですので、よくわからない方は『SX-WINDOW~』の95ページを参照してください。

要するに、どのような処理をしなければならないかといえば、ウィンドウコンテンツの内容を再描画すればよいのでした。ここでも、レフトダウンイベントのところで用意した時刻描画ルーチンを流用できそうです。

アップデートイベントに対応する処理には一定の手順があって、次のような順番で行うことが要求されています。

1) アップデート開始を宣言

2) ウィンドウコンテンツ内の再描画

3) アップデート終了を宣言

再描画を行う前後に、かならず 1) と 3) が必要です。これらの目的にはそれぞれ SX コールが用意されているので、それを呼び出すだけでよいのですが、それだけのことで忘れると、SX-SYSTEM 全体に影響を及ぼす場合があるので注意してください。1), 3) はあまりにも当然のことですから、処理を書き留めておく紙に書いておくことはないかもしれません。

2) では、例の時刻描画ルーチンを利用して、ウィンドウコンテンツ内部を全面的に書き直すことにします。『SX-WINDOW~』でも述べましたが、全体を書き直すようにしても、実際に描画されるのはアップデートリージョン内部だけですから、無駄な描画は行われません。

ところで、ウィンドウ内部には時刻の表示以外にもいろいろ表示しなければならないものがあります。コントロールや、コントロールのタイトル、そのほかの文字列や枠なども描画しなければなりません。これらの描画も時刻表示ルーチンの中に組み込んでしまうと、アイドルイベントのたびにたくさんの仕事をしなければならなくなり、SX-WINDOW 全体のスピードが低下することが考えられるので、あまりいい手ではありません。

しかし、ウィンドウコンテンツ内を描画する処理は初期化のときにも使いそうなので、できればサブルーチンにしておきたいところです。それで、時刻表示用のルーチンと、時刻以外の描画ルーチンの2つを用意して、2) ではこの2つを呼び出すことにします。2つめのサブルーチンを用意することも紙に書いておきましょう。

9 アクティベートイベント

このイベントもアップデートイベント同様、ウィンドウを持っているアプリケーションはかならず対応しなければなりません。アクティベートイベントは、デスクトップ中に存在するウィンドウのうちのどれかの位置がもっとも手前になった（アクティベートされた）場合に発生します。ウィンドウ群に上下関係の変化が起こったときに発生すると考えてもよいでしょう。

このイベントでは、アプリケーションはアクティブになったのが自分のウィンドウであるかどうかを調べ、その結果をワークエリアに記憶しておきます。このワークエリアの内容は、右クリックによるメニューの呼び出しの際などに利用されますが、私たちの時計の場合、参照することはないかもしれません。しかし、いちおう決まり文句と考えると、この処理を用意しておくことにします。

これ以外にも、たとえば、アクティブであるときとそうでないときで表示の内容を変えたいという場合などに、このイベントで処理を行うことがあります。

10 システムイベント1

11 システムイベント2

システムイベントは、SX-WINDOW 上のすべてのタスクを調停し、全体としてうまく動

作させているタスクマネージャからの指示であると考えてよいでしょう。このため、タスクマネージャイベントと呼ばれることもあります。しかし、ほかのタスクからのメッセージも含める場合は、システムイベントと呼ぶのが正しいように思えます。

システムイベント 1 とシステムイベント 2 の違いですが、前者はおもにすべてのタスクに関係するイベントで、後者はこのタスクに狙いを絞って発行させるイベントであるという点です。多くの場合、2 つのイベントをまとめて処理してしまいます。私たちの時計でも、そのようにします。

これらのイベントにはタスクマネージャイベントコードと呼ばれる数値が付随していて、その値によって意味が違ってきます。タスクマネージャイベントコードは、『SX-WINDOW〜』188 ページで一覧表になっています。この中で、すべてのアプリケーションでサポートしなければならないのは、1 の「タスクの終了」、2 の「ウィンドウのクローズ」、32 の「ウィンドウのセレクト」です。

これら 3 つについては、次のような手順で対応できそうです。

- 1) タスクマネージャイベントコードを得る
- 2) 1 ならば、アプリケーションを終了させる
- 3) 2 ならば、同様にアプリケーションを終了させる
- 4) 32 ならば、ウィンドウをアクティブにする

私たちの時計の仕様には、「SX-WINDOW 終了時に各種設定を記憶しておき、次回に起動したときに再現する」という機能がありました。このうちの「記憶」を行うのが、ここです。システムイベントの中にタスクマネージャイベントコード 31、「現在の状態をセーブ」があります。このコードをとまってシステムイベントが発生した場合、アプリケーションは自分の状態をなんらかの方法で保存して、次回に起動されたとき、再現できるように準備しなければなりません。

したがって、

- 5) 31 ならば、12 時/24 時制の設定、時報の on/off などの状態を保存する

という処理を行うことにします。具体的な方法については、プログラムを設計する際に述べることにしましょう。前回の状態を再現する仕組みについては、70 ページのコラムで解説していますので、そちらを参照されると、理解がより深まると思います。

最後にもう 1 つ。

- 6) そのほかのタスクマネージャイベントコードであった場合は関係ないので、イベント待ちに戻る

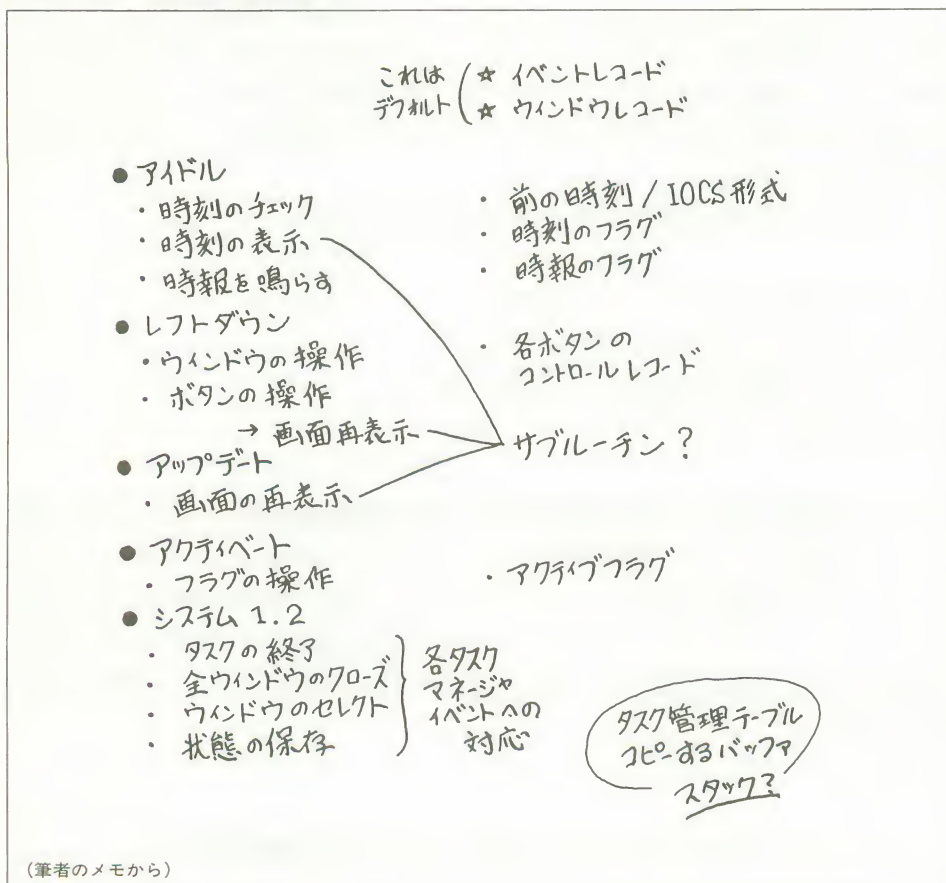
これを忘れてはいけません。

5

必要な初期化処理, 終了処理をまとめる

さて, 以上でひととおり各イベントへの対応を考えてきました。紙の上には, それぞれのイベントで行う処理の内容や, 用意すべきサブルーチン, 使用するワークエリアの表などが書かれていることと思います (図 6)。

■図 6 イベント対応処理の整理の例



これらをうまく働かせるためには, アプリケーションが起動した直後に行う初期化処理と, 終了する前に後始末をする終了処理を用意しなければなりません。

初期化処理では, 次のような仕事を行います。

● ワークエリアの初期化

紙に書いておいたワークエリアの表の中から初期化が必要なものを選び出し, 適切な初期値を代入するようにします。ワークエリアの内容はタスク起動時には不定ですから, 初期化して

いないと予想しない動作を行うことがあります。

● ウィンドウのオープン

私たちの時計もウィンドウ上で動作するのですから、当然、最初にウィンドウをオープンしておかなければなりません。コマンドライン上で '-W' オプションが指定されていた場合、アプリケーションはその位置にウィンドウを開かなければならないので、コマンドラインの解析もあわせて行う必要があります。

ウィンドウを開く際、ウィンドウレコードを作成する場所として、任意のアドレスからの領域と、ヒープゾーン中に作成した再配置不能ブロックの2種類が選べますが、メモリ効率を考えると前者とするのが正解です。ワークエリア中にウィンドウレコードを収めるための領域をあらかじめ用意しておいて、そのアドレスを指定するようにするとよいでしょう*12。

*12: はじめからウィンドウの数がわかっている場合は、このような方法をとるべきですが、動的にウィンドウの数が変化するようなアプリケーションの場合は、ヒープゾーンを利用するのもしかたがないところではあります。

● コントロールのオープン

ウィンドウ同様、コントロールもオープンしておく必要があります。先ほど紙の上に書いたウィンドウ上の情報等の配置図にしたがって、コントロールをオープンし、配置します。

● 初期画面の描画

用意しておいたサブルーチン2つを使えば、かんたんに画面全体を描画できることはもうおわかりでしょう。それぞれを呼び出して、ウィンドウ内部をすべて描画します。

おおむね、以上のようなところですが、私たちの時計の場合、もう1つすることがあります。これは、3) と 4) の間で行うとよいでしょう。

3.5) 前回の状態が保存されていれば、それを再現する

システムイベント1で、タスクマネージャイベントコード31を受け取ったときに状態を保存しました。時計が起動して初期化処理を行っているときに、かならず前回保存した「状態に関する情報」が存在しているとはかぎりませんが、保存されていた場合、その情報にしたがって状態を再現しなければなりません。私たちの時計の場合は、12時/24時制の設定と、時報のon/offでした。これらの情報にしたがってワークエリアを設定し、4)の画面の描画を行うことによって、前回の状態が完全に再現できそうです。

そして、終了処理をする場合は、

- 1) コントロールのクローズ
- 2) ウィンドウのクローズ

を行って、アプリケーションは終了することができます。メモリブロックを作成していたりする場合、ここで廃棄するようにしてください。

先ほどの紙に、初期化处理、終了処理を追加することによって、ひととおりプログラムの構成らしきものが見えてきました(図7)。これをもとにして、実際にどのようなコードを書いていくか、考えてみましょう。

■図7 初期化处理、終了処理をまとめた例

- 初期化
 - ・ワークの初期化
 - ・コマンドラインの解析 → フラグのセット
 - ・ウィンドウ開く
 - ・コントロール開く
 - ・初期画面作成 → サブルーチン
- 終了処理
 - ・コントロール閉じる
 - ・ウィンドウ閉じる

(筆者のメモより)

ある程度 SX アプリケーションの開発に慣れてきたら、これほど細々とした仕様を書かなくても、ある程度の規模までなら、頭の中だけでなんとかなる場合もあります。また、紙を使わずにエディタでテキストファイルとしてまとめあげるということもできるでしょう。ですが、矢印を引っ張ったり、マルで囲んだり、柔軟な表現が可能な紙の上で考えを整理することによって、案外すっきりしたプログラムが書けてしまうことがあることも心に留めておく必要があるでしょう。

COLUMN 用語の変更

SX-WINDOW の正式な用語と、それが定められる以前に使われていた用語が混用されているのが現状ですが、今後は正式な用語に移行していく必要があるでしょう。

ここで、旧用語と新用語を整理しておきます。

なお、本文中ではなるべく旧用語を併記するようにしています。

マウスマネージャ

・マウスカーソル → ポインタ/マウスポインタ

ウィンドウマネージャ

・クローズボックス → クローズボタン

- ・ グローボックス → サイズボタン
- ・ 矢印 → ディレクトリ戻りボタン

コントロールマネージャ

- ・ 標準ボタン → 文字ボタン
- ・ オルタネイトボタン → ラジオボタン
- ・ セレクトボタン → チェックボックス

テキストマネージャ

- ・ キャレット → カーソル

COLUMN | SX1.10 の不具合

現時点で SX1.10 には、以下のような不具合が確認されています。

なお、カッコ内は発見された方のお名前です。

- ・ \$A04C MMemAmiTpeach が正しく動作しない
- ・ \$A0DC RMRscAdd がレジスタ D3 を破壊する (中村氏)
- ・ \$A146 GMExPat でレジスタの値が異常 (沖氏)
- ・ \$A16C GMImgToRgn でテキストタイプのイメージを指定すると、右端にゴミが出る場合がある
- ・ \$A17F GMCopy で縮小が発生すると、ゴミが表示される場合がある
- ・ \$A19F GMClosePoly でディステーションがヌルリージョンの場合、不都合が発生する
- ・ \$A1A7 GMMMapRect で左下 x 座標の変換に失敗する (中村氏)
- ・ \$A1AF GMGetPixel の戻り値の上位ビットと下位ビットが逆
- ・ \$A1B2 GMCalcFrame で右端にゴミが出る場合がある
- ・ \$A1BB GMTransImg でディステーションがカレントビットマップでない場合、不都合が生じる場合がある。また、以下のビットマップ間の変換ができない。
 TXT → GRP, TXT → GR2, TXT → GR3
 GRP → TXT, GRP → GR2
 GR2 → TXT, GR2 → GRP, GR2 → GR3
 GR3 → TXT, GR3 → GR2
- ・ \$A1BF GMPaintRgn を実行する場合、ヒープゾーンに \$11000 以上の余裕がないと暴走する
- ・ \$A221 WMGetTID で正しい値が返らない (沖氏)
- ・ \$A2A3 CMUserSet で正しい値が設定できない (沖氏)
- ・ \$A2A5 CMProcSet で正しい値を設定できない (沖氏)
- ・ \$A2A7 CMDefDataSet で正しい値を設定できない (沖氏)
- ・ ダイアログアイテムとして編集可能テキストを使用できない
- ・ フォントサイズによっては表示時にゴミが表示される

沖氏(NIFTY GGC02412), 中村氏(NIFTY GBA02750), および不具合の発見にご協力いただいた皆さんに感謝いたします。

1² コードの組み立て

プログラムの仕様が決まったら、次はどのようなコードを書いてそれを実現したらよいかを考えることにします。

1 スケルトン

前節で決めた仕様について考える前に、『SX-WINDOW〜』で登場したスケルトンについて、もう一度かんたんに解説しておくことにしましょう。

SX アプリケーションを作製するうえで、「プログラミングガイドライン」と、「ユーザーインタフェースガイドライン」という2種類の決まりごとを守ることが要求されます。前者は、SX-WINDOW という OS 上のタスクとして動作するための、後者はユーザがすべてのアプリケーションを同様な方法で操作できるようにするための決まりごとです*1。

*1: ガイドラインについては、『SX-WINDOW〜』の 226 ページ以降を参照してください。

これらのガイドラインを忠実に守ろうとした場合、そのために書かなければいけないコードはかなりの量におよびます。また、いくつかアプリケーションを書いてみると、どんなアプリケーションでも似たような部分を持っていることに気がつきます。アプリケーションが起動された直後の処理や、終了するときの処理。また、ほとんどのアプリケーションはウィンドウを開くでしょうから、ウィンドウまわりの定型処理（ウィンドウのオープンから、ドラッグ、大きさの変更など）なども、この範疇です。

これらの「似たような部分」は、ほとんどのアプリケーションに最低限必要な、重要な意味を持つ部分であり、プログラム全体の成り立ちを支える骨組み——スケルトン——といえます。

SX-WINDOW にかぎらず、ウィンドウ環境のアプリケーションを書く場合は、こうした骨格をまとめて、流用可能なかたちにして用意しておくのが普通です。スケルトンという語感からもうかがえるように、実際にこれを用いて開発を行うときの感覚は、まさに「肉付けする」という言葉がふさわしいでしょう。

スケルトンだけでは何の意味もない、「ただ起動して終了するだけ」といった代物でしかありませんが、そこに目的に応じてさまざまなコードを付け加えていくことによって、アプリケーションは生き生きと動き出すのです。

まとめとして、『SX-WINDOW〜』でも挙げた、スケルトンを利用する3つのメリットを確認しておきましょう。

- 1) コードを書く量を減らすことができる
- 2) イベントドリブンを意識する必要性が少なくなる

3) プログラムのモジュール化が容易となる

『SX-WINDOW〜』に掲載した、スケルトンを構成する4つのファイルの内容を、リスト1〜4に示します。

リスト1 "SXCALL.MAC" は、SX コールを呼び出すためのマクロを定義するファイルで、SX コールを利用するソースの先頭でインクルードして使用します。

リスト2 "WORK.INC" は、ワークエリアの内容を定義するためのファイルです。ワークエリアを利用するソースの先頭でインクルードします。

リスト3 "SKELTON.S" は、スケルトンの中核をなす部分で、モジュールヘッダ、非常に下位のレベルの初期化処理、そしてイベント待ちループと、各イベント処理ルーチンへの分岐を行います。『SX-WINDOW〜』に掲載した "SKELTON.S" とは多少異なっていますが、これは SX1.10 になって SXKERNEL.X が標準添付となったことによる変更です。

リスト4 "BODY.S"*2は、"SKELTON.S" から呼ばれるルーチンの集合体で、初期化処理、各イベントの処理ルーチン、終了処理などを含みます。多くの場合、このファイルに手を加える、あるいは差し替えることによって、目的に応じたアプリケーションを作成します。

それぞれのファイルについての詳細な解説は、前著『SX-WINDOW〜』を参照してください。注釈を読んでいただければ、ある程度処理の流れを理解していただけるとと思います。

*2:『SX-WINDOW〜』で SMPLI.S として掲載したものとほとんど同じものです。ファイル名がここでは適当でないと判断したためリネームしただけで、内容はほとんど変わりありません。

■リスト1 SXCALL.MAC

```

1 *
2 *      SX-WINDOW用マクロ定義ファイル
3 *
4
5 SXCALL      macro    num          * [ SXコール呼び出しマクロ ]
6              dc.w    num
7              endm

```

■リスト2 WORK.INC

```

1 *
2 *      SX-WINDOW
3 *      サンプルプログラム # 1
4 *
5 *      ワーク定義用インクルードファイル
6 *
7
8 STKSIZE      =          2*1024          * スタックサイズ
9
10 *           ワークの内容の定義
11
12             .offset 0
13 cmdLine:
14             ds.l    1          * コマンドラインのアドレス
15 envPtr:      * を保存するワーク
                  * 環境のアドレスを

```

16		ds. l	1	* 保存するワーク
17	winRect:			* ウィンドウ
				* レクタングルレコード
18		ds. l	2	
19	paramFlg:			* コマンドラインの
				* 解析結果を示す
20		ds. w	1	* フラグ
21	eventRec:			* イベントレコードの先頭
22	eventRec_what:			* イベントコード
23		ds. w	1	
24	eventRec_whom1:			* 第1引数
25		ds. l	1	
26	eventRec_when:			* イベント発生時
27		ds. l	1	
28	eventRec_whom2:			* 第2引数
29		ds. l	1	
30	eventRec_what2:			
31		ds. w	1	* タスクマネージャ
				* イベントの種類
32	eventRec_taskID:			
33		ds. w	1	* 送り手のタスクID
34	eventMask:			* イベントマスクを保存する
				* ワーク
35		ds. w	1	
36	taskID:			* タスクIDを
				* 保存するワーク
37		ds. l	1	
38	winPtr:			* ウィンドウレコードを
				* 作成する場所
39		ds. b	\$72	
40				
41	winActive:			* アクティブフラグ
42		ds. w	1	
43				
44	WORKSIZE:			* ワークの終了
45				

■リスト3 SKELTON.S

```

1 *
2 *          SX-WINDOW
3 *          標準スケルトン
4 *
5
6          .include      DOSCALL.MAC
7          .include      SXCALL.MAC
8
9          .xref         _INIT,_TINI
10         .xref         IDLE
11         .xref         MSLDOWN,MSLUP,MSRDOWN,MSRUP
12         .xref         KEYDOWN,KEYUP
13         .xref         UPDATE,ACTIVATE
14         .xref         SYSTEM1,SYSTEM2,SYSTEM3,SYSTEM4
15
16         .include      WORK.INC          * ワークエリアの内容を
17                                     * 定義するファイル
18
19         .text
20 mdhead:
21         dc. l          'OBJR'          * [ モジュールヘッダ ]
                                         * R型モジュール

```

```

22          dc. l    0                                * プログラムエリアの
23          dc. l    _main-mdhead                    * サイズ(Xファイルの
24          dc. l    WORKSIZE+STKSIZE                * 場合意味がない)
25          dc. l    0, 0, 0                          * スタートアドレス
26                                     * オフセット
27 DiXstart:                                     * ワークエリアのサイズ
28                                     * システム予約
29                                     * コマンドラインから起動
30                                     * 場合
31                                     * ここからスタートする
32          lea      64(a1), a1
33          move. l  a1, sp
34          lea      16(a0), a0
35          sub. l   a0, a1
36          move. l  a1, -(sp)
37          pea      (a0)
38          dos      _SETBLOCK                        * 専有メモリを縮小する
39          clr. l   -(sp)
40          pea      comm(pc)
41          pea      shname(pc)
42          move. w  #2, -(sp)
43          dos      _EXEC                            * デバッグ用カーネルのパス
44                                     * をサーチする
45          clr. l   -(sp)
46          pea      comm(pc)
47          pea      shname(pc)
48          clr. w   -(sp)
49          dos      _EXEC                            * デバッグ用カーネルを
50                                     * 立ち上げる
51          tst. l   d0
52          bpl      p_execil                          * 正常に終了した場合
53                                     * そのまま終了
54          pea      mes_execerr(pc)
55          dos      _PRINT                            * エラーメッセージを
56 p_execil:                                     * 表示する
57          dos      _EXIT                            * 終了
58
59          .data
60 mes_execerr:
61          dc. b    'カーネルの起動に失敗しました!!!', 13, 10, 0
62          .even
63 shname:
64          dc. b    'SXKERNEL.X -K -R7 -L1', 0        * カーネルの名前
65          ds. b    70
66          .even
67
68          .bss
69 comm:
70          ds. b    258
71
72                                     * カーネルはここから先の
73                                     * コードを読み込み、
74                                     * タスクとして立ち上げる
75          .text
76 _main:
77                                     * SX-SHELLから
78                                     * 起動した場合
79                                     * ここからスタートする
80          movea. l  a1, a5
81          move. l   a2, cmdLine(a5)
82          move. l   a3, envPtr(a5)
83          clr. w   -(sp)
84          clr. l   -(sp)

```

```

83      pea.l   winRect (a5)
84      pea.l   (a2)
85      SXCALL  $A3EA          # __TSTakeParam
86      lea.l   14(sp), sp    # コマンドラインを解析し、
87      move.w  d0, paramFlg (a5) # '-W オプションを得る
88
89      bsr     _INIT          # アプリケーションの初期化
90      bmi     _exit          # 初期化時に
                              # エラーがあれば終了
91
92      move.w  #$ffff, eventMask (a5)
93 loop:                                # メインループ
94      pea     eventRec (a5)
95      move.w  eventMask (a5), -(sp)
96      SXCALL  $A357          # __TSEventAvail
97      addq.l  #6, sp         # イベントを得る
98      lea     eventTable(pc), a1
99      move.w  eventRec_what (a5), d0
100     and.w   #15, d0
101     add.w   d0, d0
102     move.w  (a1, d0.w), d0   # イベントコードによって
103     jsr     (a1, d0.w)      # 分岐する
104     tst.l   d0
105     bmi     _exit
106     bra     loop
107
108 eventTable:                        # 分岐先のテーブル
109     dc.w    IDLE-eventTable      # 0 アイドルイベント
110     dc.w    MSLDOWN-eventTable  # 1 レフトダウンイベント
111     dc.w    MSLUP-eventTable    # 2 レフトアップイベント
112     dc.w    MSRDOWN-eventTable  # 3 ライトダウンイベント
113     dc.w    MSRUP-eventTable    # 4 ライトアップイベント
114     dc.w    KEYDOWN-eventTable  # 4 キーダウンイベント
115     dc.w    KEYUP-eventTable    # 6 キーアップイベント
116     dc.w    UPDATE-eventTable   # 7 アップデートイベント
117     dc.w    DAMMY-eventTable    # 8 --
118     dc.w    ACTIVATE-eventTable # 9 アクティベイトイベント
119     dc.w    DAMMY-eventTable    # 10 --
120     dc.w    DAMMY-eventTable    # 11 --
121     dc.w    SYSTEM1-eventTable  # 12 システムイベント1
122     dc.w    SYSTEM2-eventTable  # 13 システムイベント2
123     dc.w    SYSTEM3-eventTable  # 14 システムイベント3
124     dc.w    SYSTEM4-eventTable  # 15 システムイベント4
125
126 DAMMY:
127     rts
128
129 _exit:
130     bsr     _TINI          # [ 終了する ]
                              # アプリケーションの
                              # 終了処理
131
132     move.w  d0, -(sp)
133     SXCALL  $A352          # __TSExit
134
135     .end    DiXstart
136

```


■リスト4 BODY.S

```

1 *
2 *      SX-WINDOW
3 *      標準スケルトン
4 *
5 *      初期化&終了&イベント処理モジュール
6 *
7
8      .include      DOSCALL.MAC
9      .include      SXCALL.MAC
10
11     .xdef      _INIT, _TINI
12     .xdef      IDLE
13     .xdef      MSLDOWN, MSLUP, MSRDOWN, MSRLUP
14     .xdef      KEYDOWN, KEYUP
15     .xdef      UPDATE, ACTIVATE
16     .xdef      SYSTEM1, SYSTEM2, SYSTEM3, SYSTEM4
17
18     .include      WORK.INC      * ワークエリアの内容
19                                * を定義するファイル
20 WINOPT      =      %0000      * ウィンドウオプション
21 WIN_X      =      256        * ウィンドウ初期x
22 WIN_Y      =      128        * ウィンドウ初期y
23
24     .text
25 IDLE:      * [ アイドルイベント ]
26 MSLUP:      * [ レフトアップイベント ]
27 MSRDOWN:    * [ ライトダウンイベント ]
28 MSRLUP:     * [ ライトアップイベント ]
29 KEYDOWN:    * [ キーダウンイベント ]
30 KEYUP:      * [ キーアップイベント ]
31 SYSTEM3:    * [ システムイベント3 ]
32 SYSTEM4:    * [ システムイベント4 ]
33     moveq     #0, d0          * 以上のイベントでは
34                                * なにもしない
35     rts
36 MSLDOWN:    * [ レフトダウンイベント ]
37     move.l    eventRec_whoml(a5), a0
38     lea       winPtr(a5), a2
39     cmp.l     a2, a0          * 自分のウィンドウ上で
40     bne       MSLD9          * 発生したか?
41                                * 違うならMSLD9へ
42     tst.b     winActive(a5)
43                                * 現在ウィンドウは
44                                * アクティブか?
45                                * アクティブならMSLD1へ
46     bne       MSLD1
47     pea       (a2)
48     SXCALL    $A1FE          * __WMSelect
49     addq.l    #4, sp
50     bra       MSLD9          * アクティブにするだけ
51 MSLD1:
52     pea       eventRec(a5)
53     pea       (a2)          * ウィンドウ処理
54     SXCALL    $A3A2          * __SXCallWindM
55     addq.l    #8, sp
56     tst.l     d0            * どこも操作されなかった?
57     beq       MSLD9          *   ならばMSLD9へ
58     cmp.w     #7, d0          * クローズボタン?
59     beq       CloseBttn      *   ならばCloseBttnへ
60 MSLD9:

```

```

59          moveq    #0, d0
60          rts
61
62 CloseBttn:
63          moveq    #-1, d0
64          rts
65
66 UPDATE:
67          pea      winPtr(a5)
68          SXCALL   $A20D
69          addq.l   #4, sp
70
71          bsr      DrawGraph
72
73          pea      winPtr(a5)
74          SXCALL   $A20E
75          addq.l   #4, sp
76
77          moveq    #0, d0
78          rts
79
80 ACTIVATE:
81          move.l   eventRec_whoml(a5), d0
82          beq      ACT9
83          lea      winPtr(a5), a0
84          cmp.l    a0, d0
85          bne      ACT0
86          st       winActive(a5)
87
88          bra      ACT9
89
90 ACT0:
91          sf       winActive(a5)
92
93
94 ACT9:
95          moveq    #0, d0
96          rts
97
98 SYSTEM1:
99 SYSTEM2:
100          move.w   eventRec_what2(a5), d0
101          cmp.w    #1, d0
102          beq      AllClose
103          cmp.w    #2, d0
104          beq      AllClose
105          cmp.w    #$20, d0
106          beq      WindowSelect
107
108          moveq    #0, d0
109          rts
110
111 AllClose:
112          moveq    #-1, d0
113          rts
114
115 WindowSelect:
116          pea      winPtr(a5)
117          SXCALL   $A1FE
118          addq.l   #4, sp
119
120          moveq    #0, d0
121          rts

```

* [アップデートイベント]

* __WMUpdate
* アップデート開始

* ウィンドウ内部を描画

* __WMUpdtOver
* アップデート終了

* [アクティベートイベント]

* 自分のウィンドウが
* アクティブになった?
* 違うのならACT0へ
* アクティブフラグを
* セット

* アクティブフラグを
* リセット

* [システムイベント1]
* [システムイベント2]

* タスクの終了?
* ならばLetsGoAwayへ
* 全ウィンドウのクローズ?
* ならばLetsGoAwayへ
* ウィンドウのセレクト?
* ならばWindowSelectへ

* 自分のウィンドウを
* セレクトする
* __WMSelect

```

119 _INIT:                                     * [ アプリケーション
                                           * の初期化を行なう ]

120      move.l winRect(a5), d0
121      move.w paramFlg(a5), d1
122      btst    #0, d1                        * '-W オプションが
                                           * 指定された?
                                           * 指定されていないければ
                                           * _INIT0へ

123      beq     _INIT0

124
125      move.l winRect+4(a5), d1              * 正しいレクタングルが
126      beq     _INIT1                      * 指定されたかどうか
                                           * を調べる

127      tst.w   d1
128      cmp.w   d0, d1
129      ble     _INIT1
130      swap    d0
131      swap    d1
132      cmp.w   d0, d1
133      bgt     _INIT2
134      swap    d0
135      swap    d1
136      bra     _INIT1

137 _INIT0:
138      SXCALL  $A35E                        * _ISGetWindowPos
139      move.l  d0, winRect(a5)              * デフォルト位置を得る

140 _INIT1:
141      add.l   #WIN_X*$10000+WIN_Y, d0      * ウィンドウレクタングルを
                                           * 作成

142      move.l  d0, winRect+4(a5)

143 _INIT2:
144      SXCALL  $A360                        * _ISGetID
145      move.l  d0, taskId(a5)              * タスクIDを得る

146
147      move.l  d0, -(sp)                    * タスクID
148      move.w  #-1, -(sp)                   * クローズボタン?
149      move.l  #-1, -(sp)                   * ならばCloseBtnへ
150      move.w  #20*16+WINOPT, -(sp)        * 標準ウィンドウ
151      move.w  #-1, -(sp)                   * 可視
152      pea.l   winTitle(pc)                 * ウィンドウタイトル
153      pea.l   winRect(a5)                  * ウィンドウレクタングル
154      pea     winPtr(a5)                   * ワーク上に作成
155      SXCALL  $A1F9                        * _WMOpen
156      lea.l   26(sp), sp                  * ウィンドウを開く
157      tst.l   d0                           * エラー?
158      bmi     _INIT_Err                    * ならば_INIT_Errへ
159      st       winActive(a5)               * アクティブフラグを
                                           * セット

160
161      bsr     DrawGraph1st                 * ウィンドウ内部を
                                           * 描画する
                                           * (最初の1回)

162
163      moveq   #0, d0
164      rts
165 _INIT_Err:
166      moveq   #-1, d0
167      rts
168
169 DrawGraph1st:
170
171      rts                                    * ウィンドウ内部の描画の
                                           * 準備をするサブルーチン
                                           * (なにもしない)

172
173 DrawGraph:
174
175      rts                                    * ウィンドウ内部を描画する
                                           * サブルーチン
                                           * (なにもしない)

```

```

176
177
178 _TINI:                                # [ 終了処理 ]
179         pea    winPtr(a5)            # ウィンドウをクローズする
180         SXCALL  $A1FB                 # WMClose
181         addq.l  #4, sp                # WMDisposeでないことに注意
182
183         moveq   #0, d0
184         rts
185
186         .even                                # [ 固定データ ]
187 winTitle:
188         dc.b    7, 'NOTITLE'            # ウィンドウタイトル
189
190         .end

```

A>AS SKELTON

A>AS BODY

A>LK -O SAMPLE SKELTON BODY

のようにして SKELTON.S と BODY.S をそれぞれアセンブルし、リンクすることで、ウィンドウを表示するだけの基本的なプログラム "SAMPLE.X" が得られます。このスケルトンは、今後何度も使い回すことになるのですから、きちんと動作するかどうかを確認しておいてください。ウィンドウのドラッグ、アクティベート、そしてクローズボタンによるタスクの終了が正常に行えれば、いちおう正常に動作しているといえます。

ところで、SX アプリケーションのプログラム＝プログラム・モジュールには "R 型"、"C 型"、"O 型" の 3 つの種類がありました。このスケルトンは R 型を前提につくられています。R 型のモジュールはコードを複数のタスクで共有するため、メモリ効率が非常によいのが特徴です*3。

*3:R 型のモジュール実行時のメモリの使われ方については、「SX-WINDOW～」の 192 ページ以降を参照してください。

そのために、プログラムは次の 2 点を守って書くことが要求されます。

- 1) 変数などは必ずワークエリアの中に置く
- 2) コード部、固定データ部の内容を自分で書き換えない

1) に関しては、仕様をまとめた紙に書き出した変数を、原則としてワークエリアの中に置くようにすればよいでしょう。スケルトンでは、最初の初期化時にワークエリアの先頭アドレスを A5 に収めているので、以降 A5 をポインタとして、ワークエリアへアクセスできます。WORK.INC にはすでに基本的な変数が定義してありますが、たとえば、この中の変数 taskID の内容を D0 に読み込みたい場合は


```
move.l    taskID(a5),d0
```

と書けばよいのです。同様に、イベントレコード eventRec のアドレスを A0 にセットしたい場合は、

```
lea       eventRec(a5),a0
```

とします。

2) に関しては、とくに付け加えることはありません。コード部、固定データ部はプログラムエリアを共有するすべてのタスクが利用するので、かつてに書き換えれば、問題が発生することは明らかです*4。

*4:発生する問題を逆手にとって、プログラム部を共有するタスクすべてにある情報を伝えたい場合など、静的なワークを利用して伝えることもできます。ただし、プログラム部の共有のしくみをよく理解したうえでなければ利用することはおすすめできません。

2 仕様からコードを起こす

スケルトンの用意ができたところで、私たちの時計の話に戻しましょう。

前節では、仕様を決めるにあたって大きく次の3つに分けて行いました。これらは、それぞれ BODY.S の中に収められたルーチンを書き換える、または追加することで実現できます。対応関係は次のとおりです。

- ・ウィンドウ内部の構成
 - ウィンドウコンテンツへの描画処理ルーチン
- ・各イベントへの対応
 - 各イベントの処理ルーチン
- ・初期化/終了処理
 - 初期化処理/終了処理ルーチン

スケルトンを利用することによってウィンドウを出すところまではできあがっていますから、これらのルーチンに手を加える程度ですむことはもうわかりですね。

では、これらのルーチンへの手の加え方について、順に解説していくことにします。

①ウィンドウコンテンツへの描画処理ルーチン

BODY.S では、ウィンドウコンテンツへの描画は 173 行目からのサブルーチン、DrawGraph にまとめてあります。いまのところはウィンドウを出すだけで描画は行っていないので、何もせずにリターンしていますが、ここに描画用のコードを書いておけば、画面を描

く、あるいは描き直す必要が生じたときに呼び出されることになります。いまのところ、アップデートイベントが発生した際に画面を再描画するために呼び出すようになっていますが、前節でも述べたように画面を描き直さなければならない場面はいくつも考えられます。そのたびに DrawGraph を呼び出せば適切な描画が行えるようにしておかなければならないでしょう。

私たちの時計の場合、その時々によって必要な場所だけを再描画できるようにするため、2つのルーチンを用意することになっていました。時刻表示用のルーチンと、時刻以外の描画ルーチンの2つです。BODY.S の DrawGraph はアップデートイベントで呼ばれていることからわかるように、ウィンドウコンテンツ内を全面的に描画するためのルーチンですが、これを DrawTime と DrawOhter の2つのルーチンに分けることにします。DrawGraph を呼び出しているところは、この2つを順に呼び出すように書き換えることにしましょう。

1) 時刻表示用ルーチン DrawTime

ウィンドウコンテンツ内への描画を始めるときに忘れてはならないことにカレントグラフポートの設定があります。これを忘れると、まったく関係のない場所に描画が行われたりすることになりますから、描画ルーチンの先頭などで設定しておくとういでしょう*5。一般に、カレントグラフポートを設定する際に指定するグラフポートレコードへのポインタとしては、ウィンドウレコードへのポインタで代用します。これが可能な理由は、ウィンドウレコードの構造を考えていただければ自明でしょう。

*5:少なくとも、タスクが切り替わる可能性のある SX コールを利用した後で描画を行う前には、必ずカレントグラフポートを設定しておく必要があります。それ以降は、次にタスクが切り替わるかもしれない SX コールを利用するまでは設定し直す必要はありませんが、何度も設定を行う分には何の問題もありません。

私たちの時計のウィンドウのウィンドウレコードを、ワークエリア内の winPtr から作成することになると、時刻表示用ルーチンは次のような始まり方をすることになります。

DrawTime:

```

    pea      winPtr(a5)
    SXCALL   $A131          * GMSetGraph
    addq.l   # 4,sp

```

このルーチンの目的は時刻を文字列として表示することですから、どうにかして現在時刻を得る必要があります。前節のアイドルイベントの項で述べたように、現在時刻は IOCS を呼び出すことによって得られます。

```

    moveq    # $56,d0      * _TIMEGET
    trap     # 15          * IOCS 呼び出し

```

これによって得られる数値は直接文字列に変換できないので、もう一度 IOCS を利用して形式を変換します。

```

move.l    d0,d1
moveq     # $57,d0          * _TIMEBIN
trap      # 15              * IOCS 呼び出し

```

この結果、D0 の内容は、上位ワードの時の位、下位ワードの上位バイトに分の位、下位バイトに秒の位が入ります。

今度はこれを文字列に直さなければなりません。つねに 24 時制で表示する時計であれば話はんたんなのですが、私たちの時計は 24 時制で表示するか 12 時制で表示するか、また、12 時制の場合は午前か午後かで表示する内容が変わってきます。このあたりの条件判断を行って、データを少し加工することにします。

```

move.w     #' ',d7          * 12 時制の場合、後で AM/PM の文
                             * 字列を入れる
swap       d0              * 上位ワードのときの位を下位に

tst.w      jisei12(a5)      * 12 時制かどうかのフラグ
beq        DrawTime2       * 24 時制なら DrawTime2 へ

cmp.w      # 12,d0          * 12 時以降？
bcc        DrawTime0       *   ならば午後なので DrawTime0
                             *   へ

move.w      #'AM',d7        * 午前
bra        DrawTime1

DrawTime0:
move.w      #'PM',d7
sub.w      # 12,d0          * 12 時間引く

DrawTime1:
tst.w      d0              * 0 時？
bne        DrawTime2       *   でなければ DrawTime2 へ

move.w      # 12,d0         * 12 時に直す

DrawTime2:
swap       d0              * 上位/下位ワードを元の順に戻す

```

加工済みのデータを IOCS の \$5B を使って文字列にします。時刻の文字列を作成するバッ

ファには、ワークエリアに用意しておいた時刻文字列を収めるためのバッファ timeStr を指定します*6。

*6:一時的にしか使わないバッファですから、スタックフレーム上に link 命令等で確保したほうがメモリ効率が上がります (ただだか数バイトですが)。余力のある方は、そのように改造してみてください。

move.l	d0,d1	*D1:時刻
lea	timeStr(a5),a1	*A1:文字列が返るバッファへのポインタ
moveq	#\$5b,d0	*_TIMEASC
trap	#15	*IOCS 呼び出し

以上で表示すべき文字列は得られました。時刻の表示を行う準備として、時刻を表示する枠の内部に現在表示されているものを、背景色で塗り潰すことによって消しておきます。塗り潰す四角形を表現するレクタングルレコードは、固定データ timeRectInside として用意しておいたものを指定することにします。

move.w	#\$100,-(sp)	*バックグラウンドカラーで描画
SXCALL	\$A144	*GMPenMode
addq.l	#2,sp	
pea	timeRectInside(pc)	*塗り潰すべき四角形の内側
SXCALL	\$A173	*GMFillRect
addq.l	#4,sp	

時刻を描画する際のフォントについて設定しておきましょう。原則として、グラフィポート内のフォントやペンの設定などは一度行っておけば変化することはないので、何度も行う必要はないはず。しかし、ほかの表示を行う際に変更している可能性があるのと、これから行う描画の内容を明示する意味で、あえて設定することにします。仕様にしたがって、フォントカインド、フォントフェイス、フォントモード、フォアグラウンドカラーの設定を行います*7。

*7:バックグラウンドカラーについては、ほかのルーチンによっても変更されることはない判断し、設定を行いません。

また、背景色で塗り潰したときに変更したペンモードを設定していないのは、文字列の描画にペンモードは影響しないからです。

move.w	#2,-(sp)	*24×24
SXCALL	\$A18B	*GMFontKind
addq.l	#2,sp	
move.w	##00011,-(sp)	*イタリック+ボールド


```

SXCALL  $A18C                                * GMFontFace
addq.l   # 2,sp
move.w   # 0,-(sp)                            * pset
SXCALL  $A18D                                * GMFontMode
addq.l   # 2,sp
move.w   # 11,-(sp)                          * 黒
SXCALL  $A147                                * GMForeColor
addq.l   # 2,sp

```

そして、先ほどつくった時刻文字列を描画します。描画を始める位置は (24, 8) あたり、ということになっていました。

```

move.l   # $0018_0008,-(sp)                  * (24, 8)
SXCALL  $A16E                                * GMMove
addq.l   # 4,sp
pea      timeStr(a5)                          * 時刻文字列のバッファ
SXCALL  $A192                                * GMDrawStrZ
addq.l   # 4,sp

```

さらに、フォントの設定を行った後、"AM", "PM", または空白も描画します。D7 に収めてある文字列を表示するために、多少変則的にスタックを利用してみることにします。

```

move.w   # 0,-(sp)                            * 12×12
SXCALL  $A18B                                * GMFontKind
addq.l   # 2,sp
move.w   # %00001,-(sp)                      * ボールド
SXCALL  $A18C                                * GMFontFace
addq.l   # 2,sp

move.l   # $0088_0014,-(sp)                  * (136, 20)
SXCALL  $A16E                                * GMMove
addq.l   # 4,sp

swap     d7
clr.w    d7                                  * これで D7 は 'AM', 0, 0 という形式になる
move.l   d7,-(sp)                            * スタックフレーム上に ASCIIZ 文字列として置く
pea      (sp)                                * スタックフレーム上の文字列を指す

```

```

    SXCALL    $A192                *GMDrawStrZ
    addq.l    #8,sp

```

以上で時刻の表示は完了です。最後にサブルーチンからのリターンを行い、このサブルーチンは終了ということになります。

```

    rts                *of DrawTime

```

2) 時刻以外の描画ルーチン DrawOther

2つの画面描画用のサブルーチンのもう一方は、時刻以外の描画を行うルーチンです。時刻以外といっても漠然としているので、最初に整理しておくことにしましょう。

- ・時刻表示を囲む枠の描画
- ・コントロールの描画
- ・コントロールのタイトルの描画

以上の3つが、このルーチンのおもな仕事です。順にコードを書いていきましょう。

まず、なによりも先にカレントグラフポートの設定を行います。実際には、ここでは設定を省略することができます。すでに DrawTime で設定していますし、このルーチンが呼ばれるときには、必ずその前に DrawTime が呼ばれているからです。しかし、何度設定を行っても悪いことはありませんし、描画開始時には必ずカレントグラフポートを設定するという習慣をつけるためにも、きちんと設定しておくことにします。

DrawOther:

```

    pea      winPtr(a5)
    SXCALL    $A131                *GMSetGraph
    addq.l    #4,sp

```

まずは最初の仕事、時刻表示を囲む枠の描画を行います。描画する枠を表現するレクタングルレコードは、固定データ timeRectOutside を指定することになります。DrawTime の中で使用した timeRectInside というレクタングルレコードを流用できそうな気もしますが、そうすると背景色での塗り潰しを行ったときに枠まで塗り潰されてしまうので、わずかな違いではありますが、別々のレクタングルレコードとしました。

この枠は影付きの枠なので、\$A1A2 GMShadowRect で描画します。このコールではペン関係の設定は参照されず、つねに一定の形式で描画されるので、ペンモードなどの設定を省略できます。

```

pea      timeRectOutside(pc)
SXCALL   $A1A2                      * GMShadowRect
addq.l    # 4,sp

```

次のコントロールの描画ですが、これはかんたんで、描画したいコントロール(複数)の乗ったウィンドウのウィンドウレコードを指定して\$A28E CMDraw を呼び出せば、そのウィンドウ上のコントロールがすべて描画されます。

```

pea      winPtr(a5)
SXCALL   $A28E                      * CMDraw
addq.l    # 4,sp

```

3 つめはコントロールのタイトル類の表示です。これらはひたすら文字列を描き始める座標を指定して、タイトル文字列を描画 (\$A1A1 GMShadowStrZ) するだけなので、例として示すのは、時報を設定するチェックボックスのタイトル 1 つだけにしておきます。

```

move.w    # 0,-(sp)                  * 12×12
SXCALL     $A18B                      * GMFontKind
addq.l     # 2,sp
move.w    # %00000,-(sp)             * 装飾なし
SXCALL     $A18C                      * GMFontFace
addq.l     # 2,sp

move.l     # $0054_0040,-(sp)         * (84, 64)
pea        chkBoxTitle(pc)            * チェックボックスのタイトル文字列 (ASCIIZ)
SXCALL     $A1A1                      * GMShadowStrZ
addq.l     # 8,sp

```

同様にして、すべてのタイトルを描画し終わったら、DrawOther の仕事はすべて終了です。

```

rts                      * of DrawOther

```

② 各イベントの処理ルーチン

私たちの時計でサポートするイベントは、次の 5 種類 6 イベントで、それぞれのイベントとイベント処理ルーチンとの対応は、次のとおりです。

- ・アイドルイベント
→IDLE
- ・レフトダウンイベント
→MSLDOWN
- ・アップデートイベント
→UPDATE
- ・アクティベートイベント
→ACTIVATE
- ・システムイベント 1, 2
→SYSTEM1
SYSTEM2

BODY.S のはじめのほうでは、サポートしないイベントの処理ルーチンのラベルをまとめて書いておき、何もせずにリターンさせている箇所があります。ここには私たちの時計のサポートするアイドルイベントの処理ルーチン `IDL` が含まれているので、25 行目の

IDLE: * [アイドルイベント]

を削除して、別の場所にアイドルイベント処理ルーチンを書くことになります。

BODY.S の中に含まれるイベント処理ルーチンは、SKELTON.S 中のイベント待ちループからサブルーチンとして呼び出されます。各イベント処理ルーチンの中で何か異常（あるいは仕様で決めておいた事態）が発生して、タスクを終了しなければならなくなった場合、DO に負の数を、正常に終了した場合は DO に 0 を入れてリターンする約束になっています。それでは、順に各イベント処理ルーチンの作成、書き換えについて考えてみましょう。

1) アイドルイベント処理ルーチン IDLE

このルーチンはいままで用意されていませんでした。MSLDOW の前あたりに置くことにします。

前節で定めた仕事の内容を順にコードに直していくことにしましょう。

最初の仕事は、「イベントレコードからイベントの発生した時刻を得る」でした。イベントの発生したシステム時刻はイベントレコードの 6 バイト目、スケルトンの定義ではワークエリア内の `eventRec when` からロングワードで収められています。

```
IDLE:
    move.l    eventRec when(a5),d1    *イベント発生時刻をD1に
```


次は、「前回時刻表示を書き換えたときに保存しておいた時刻と比較して、1 秒以上経過しているかどうかを調べる」です。これはただの計算なので説明の必要はないと思います。前回時刻を書き換えたシステム時刻はワークエリア内の lastTimeUpdate に収められていることになっていますから、コードは次のようになります。

```

move.l    d1,d0                * 前回の書き換えシステム時刻からの
sub.l     lastTimeUpdate(a5),d0 * 経過時間を求める
cmp.l     #100,d0              * 1 秒以上経過？
bcs       IDLE9                *   でなければ IDLE9 へ

```

IDLE9 はアイドルイベント処理ルーチンの終端であり、イベント待ちループへ戻る RTS 命令が書いてあります。

1 秒以上経過していると判断できた場合は、前の時刻を塗り潰し、現在時刻を描画します。このためのルーチンは、すでに DrawTime としてつくってあるので、これを呼び出すだけですみます。DrawTime を呼び出す前に、本来は次のステップとしていましたが、時刻を描画し直した時刻を変数に保存しておくことにしましょう。これは、D1 に保存しておいたイベント発生システム時刻が DrawTime 内で破壊される可能性があるためです。

```

move.l    d1,lastTimeUpdate(a5) * イベント発生時刻を保存
bsr       DrawTime

```

時刻が描画できたら、次は時報の処理です。

最初に時報を鳴らすか鳴らさないかを調べます。時報を鳴らす/鳴らさないの設定は、ワークエリア内の jihou という変数に収められています。

```

tst.w     jihou(a5)            * 時報を鳴らす？
beq       IDLE9                *   鳴らさないなら IDLE9 へ

```

時報は毎正時、つまり、00 分 00 秒に鳴らすわけですから、時刻を意味する数値の下位ワードを調べればすぐにわかります。

```

moveq     #56,d0               * _TIMEGET
trap      #15                  * IOCS 呼び出し
tst.w     d0                   * 00 分 00 秒？
bne       IDLE9                *   でなければ IDLE9 へ

```

正時であったことが確認できたところで、時報の音を出します。ここでは簡易な方法を選び、ビーブ音を鳴らすだけとします。

```

move.w    # 2, -(sp)          * 1 回
SXCALL    $A2D7               * __DMBeep
addq.l    # 2, sp

```

余力のある方は、自前の ADPCM データを鳴らしたり、予告音を鳴らすようにするなど改造してみるものよいでしょう。

以上で、アイドルイベントで行う仕事は完了しました。正常終了を意味する 0 を DO に収めて、RTS 命令でイベント待ちに戻ります。ここにラベル IDLE9 を設定することを忘れてはいけません。

```

IDLE9:
    moveq   # 0, d0
    rts                    * of IDLE

```

2) レフトダウンイベント処理ルーチン MSLDOWN

手を加えていない状態のスケルトンは、レフトダウンイベント発生時にはごく基本的な仕事をしてくれます。まず、自分のウィンドウ上でマウスの左ボタンが押されたのかどうかを判断し、ウィンドウ上であれば、ウィンドウを利用するアプリケーションすべてが守らなければならないガイドラインにそった仕事を行います（前節参照）。

スケルトンはガイドラインに示された仕事を、さまざまな条件を判断して行うようにつくられています。そのほとんどは BODY.S の 51 行、

```

SXCALL    $A3A2               * __SCallWindM

```

この 1 行に集約されています。ただし、私たちの時計の場合は、その恩恵にあずかるのはウィンドウのドラッグ程度ですが。

SXCallWindM から戻ってきたとき、DO にはマウスの左ボタンによって「押された」ウィンドウのパートコードが入っています。このパートコードは、すでに「処理された」パートを意味していることに注意してください。たとえば、パートコードとして 7 が返ってきたとき、それはクローズボタン上で左ボタンが押され、そして、やはりクローズボタン上で左ボタンが離された、ということの意味をしています*8。スケルトンは、返り値のパートコードを調べてクローズボタンであった場合は、終了すべきであることを示す負の数（-1）を DO に入れて、イベント待ちループへと戻っています。

*8: SXCallWindM 同様に、マウスポインタの座標を渡すと、ウィンドウのパートコードを返す SX コールに WMFind がありますが、WMFind は、ただその座標がウィンドウ上のどこに位置するかを調べるだけで、本文中で例に挙げたようにクローズボタン上で押され、離されたかどうかなどを調べたりすることはありません。

SXCallWindM と WMFind の返すパートコードの違いについては、『SX-WINDOW〜』178 ページのコラムも参照してください。

このように、OS と関係しているウィンドウの枠などに関する処理はスケルトンが行ってくれるので、私たちはアプリケーションの領分であるところのウィンドウコンテンツ内部の処理だけを考えればよいのです。したがって、スケルトンに手を加えるところは、スケルトンが必要なパートコードを調べ終わったところ、つまり BODY.S の 56 行、

```
cmp.w      # 7,d0      *クローズボタン?
beq        CloseBtn    *   ならば CloseBtn へ
```

の直後からということになります。

まずはマウスの左ボタンが押されたのが、ウィンドウコンテンツの内部であるかどうかを確かめなければなりません。ウィンドウコンテンツのパートコードは 3 ですから、それ以外だった場合はイベントループに戻るようにします。

```
cmp.w      # 3,d0      *ウィンドウコンテンツ?
bne        MSLD9       *   でなければ MSLD9 へ
```

私たちの時計には、ウィンドウコンテンツ内に置かれ、マウスで操作できるものとしてはチェックボックスが 1 つとラジオボタンが 2 つありました。これらは、いずれもコントロールです。ウィンドウコンテンツに置かれたコントロールについての処理を行うには、\$A3A3 SXCallCtrlM という便利な SX コールが使えます。

SXCallCtrlM は、マウスポインタの座標を調べて、ウィンドウコンテンツ上のコントロールのうちのいずれかの上で左ボタンが押されている *9 ときには、次のような処理を行ってくれます。

*9: いずれも、SXCallWindM のクローズボタンの例のように、「押されて、離される」までを確認した後処理を行います。

- ・ラジオボタン/チェックボックスの on/off
- ・スクロールバーによるスクロール
- ・そのほかのコントロールの操作

そして、処理を行ったコントロールへのハンドルを AO に、押されていたコントロールの

パートコードを D0 に返します。

このコールを呼ぶことによって、かなりの処理を SX-SYSTEM が肩代わりしてくれるので、書くべきコードは激減することがおわかりいただけると思います。結局、私たちの書かなくてはならないコードは、

- ・チェックボックスが押されたなら

押された結果 (on/off 状態) を変数に記録する

- ・ラジオボタンのどちらか片方が押されたなら、押されたボタンを on, もう片方を off にして、どちらが押されたのかを変数に記録する *10。そして、設定を変更した状態の時刻を再描画する

*10: SXCallCtrlM は、押されたラジオボタンの on/off を反転してしまうので、結局、自分で on/off を正しく設定しなくてはなりません。

これだけになります。どのボタンも押されなかった場合、D0 には 0 が返るので、この場合は何もせずにイベント待ちループに戻ることも忘れてはいけません。

それでは、実際にコードを書いてみましょう。

まずは、SXCallCtrlM を呼び出します。ウィンドウにスクロールバーがついている場合は、それへのハンドルを指定しなければならないのですが、私たちの時計のウィンドウにはスクロールバーがついていないので、省略を意味する "0" を指定します。

clr.l	-(sp)	* dRectPtr (省略)
clr.l	-(sp)	* ctrlHdlH (省略)
clr.l	-(sp)	* ctrlHdlV (省略)
pea	eventRec(a5)	* イベントレコードへのポインタ
pea	(a2)	* ウィンドウレコードへのポインタ
SXCALL	\$A3A3	* __SXCallCtrlM
lea	20(sp),sp	

押された場所がどのコントロール上でもなかった場合、イベント待ちに戻ります。

tst.l	d0	* コントロール上?
beq	MSLD9	* でなければ MSLD9 へ

続いて、押されたコントロールの種類を調べ、それにしがつて書くコントロール用の処理へ分岐します。


```

cmp.l    chkBoxHdl(a5),a0      *チェックボックスが操作された?
beq      MSLD_ChkBox          * ならば MSLD_ChkBox へ
cmp.l    rad1Hdl(a5),a0      *ラジオボタン 1 が操作された?
beq      MSLD_Rad1           * ならば MSLD_Rad1 へ
cmp.l    rad2Hdl(a5),a0      *ラジオボタン 2 が操作された?
beq      MSLD_Rad2           * ならば MSLD_Rad2 へ

```

これら以外ということはありませんが、その場合は何もしないことにしておきます。このような、「まず起こらない」エラー対策も、手を抜かずにできるだけ用意しておきたいのです。

```

bra      MSLD9                *これら以外の場合は MSLD9 へ

```

チェックボックスの処理は、MSLD_ChkBox から始まります。ここでの処理は、チェックボックスの新たな値を調べて、それを変数に格納することです。チェックボックスの値は原則として 0 (off) か 1 (on) と決まっているので、on か off かは返り値を、ワードで tst すれば判断できます。したがって、この値を変数 jihoui に代入しておけば、アイドルイベント処理ルーチンの中で時報を鳴らすか鳴らさないかの判断を行うことができます。

```

MSLD_ChkBox:
move.l    chkBoxHdl(a5),-(sp)  *チェックボックスへのハンドル
SXCALL    $A291               * __CMValueGet
addq.l    #4,sp
move.w    d0,jihoui(a5)
bra      MSLD9                *MSLD9 へ

```

ラジオボタンは 1 が on になった場合は 12 時制、2 が on になった場合は 24 時制ですから、フラグ jisei12 をセット、あるいはリセットして、押されたボタンを on、もう片方のラジオボタンを off にする処理を行います。

```

MSLD_Rad1:
moveq     #1,d1               *ラジオボタン 1 を on に
moveq     #0,d2               *ラジオボタン 2 を off に
bra      MSLD_Rad2
MSLD_Rad2:
moveq     #0,d1               *ラジオボタン 1 を off に

```

```

        moveq    # 1,d2                * ラジオボタン 2 を on に
MSLD_Rad:
        move.w   d1,jiseil2(a5)        * jiseil2 に値を設定

        move.w   d1,-(sp)
        move.l   rad1Hdl(a5),-(sp)      * ラジオボタン 1 に値を設定
        SXCALL   $A290                  * __CMValueSet
        addq.l   # 6,sp
        move.w   d2,-(sp)
        move.l   rad2Hdl(a5),-(sp)      * ラジオボタン 2 に値を設定
        SXCALL   $A290                  * __CMValueSet
        addq.l   # 6,sp
        bsr      DrawTime               * 切り替えられた時制で時刻を再描画
        bra      MSLD9

```

レフトダウンイベントの処理は以上です。

3) アップデートイベント処理ルーチン UPDATE

BODY.S のアップデート処理ルーチンに書かれているコードだけでも、私たちの時計に必要な処理の 3 分の 2 がすんでいます。したがって、手を加えなければならないのは、画面全体を書き直す描画ルーチンの呼び出しだけということになります。

BODY.S で呼び出している描画ルーチンは DrawGraph となっていますが、私たちの時計では、これを 2 つに分割し、DrawTime と DrawOther に分けました。したがって、UPDATE 全体は次のように書き換えます。

```

UPDATE:                                     * [アップデートイベント]

        pea      winPtr(a5)
        SXCALL   $A20D                    * __WMUpdate
        addq.l   # 4,sp                    * アップデート開始

        bsr      DrawTime
        bsr      DrawOther

        pea      winPtr(a5)
        SXCALL   $A20E                    * __WMUpdtOver
        addq.l   # 4,sp                    * アップデート終了

        moveq    # 0,d0
        rts

```

4) アクティブイベント処理ルーチン ACTIVATE

アクティブイベントが発生した場合の処理は、BODY.S そのままで問題ありません。このままで手を加える必要はありません。

5) システムイベント 1, 2 処理ルーチン SYSTEM1, SYSTEM2

スケルトンには、すべてのアプリケーションがサポートしなければならない、タスクマネージャイベントコード 1 の「タスクの終了」、2 の「ウィンドウのクローズ」、32 の「ウィンドウのセレクト」の 3 つについての処理が用意されています。私たちの時計には、さらにこれらに加えて、タスクマネージャイベントコード 31, 「現在の状態をセーブ」についての処理が必要です。

アプリケーションの状態の保存を行う方法としては、状態を数値化してファイルなどに記録しておくことが考えられます。しかし、私たちの時計の場合、保存すべき状態といっても、「12 時制か 24 時制か」と「時報を鳴らすか鳴らさないか」の 2 つくらいしかありません。このようなわずかな情報を収めたファイルを作成するのはディスクの効率もよくありませんし、ディスク上にファイルがどんどん増えてしまうのも考えものです。さらに、いくつも起動できるアプリケーションの場合は、ファイル名を工夫するなどの必要もあるでしょう^{*11}。

*11: 状態をファイルに記録するアプリケーションとして、SXL.I のエディタ X があります。エディタ X も、いくつも同時に起動することができますが、ファイルから最初に起動されたエディタ X が「親」となり、そのコードを共有している「子」たちの情報もまとめてファイルとして管理しているようです。

そこで、「サウンド.X」や「キャンパス.X」などでも使われている方法を使うことにします。

アプリケーションがタスクとして起動される際にはコマンドライン文字列が渡されますが、シェルを終了するときには動作していたタスクに関しては、そのタスクの起動時に渡されたコマンドライン文字列がタスク名とともに SYSDTOP.SX に記録されます (144 ページのコラム参照)。そして、次にシェルが起動されたとき、SYSDTOP.SX に記録されていたタスクが再起動され、そのときにコマンドライン文字列も記録されていたものが渡されます。この機能をうまく利用することによって、状態の保存を行うことができます。

先ほども挙げたように、私たちの時計の保存すべき状態は、「12 時制か 24 時制か」と「時報を鳴らすか鳴らさないか」の 2 つです。コマンドライン上で次のようなオプションを指定することによって、これらの初期設定を決定するように決めておきます。

-t0	24 時制
-t1	12 時制
-j0	時報を鳴らさない
-j1	時報を鳴らす

初期化ルーチンでコマンドラインを解析するときに、これらのオプションを判断するように

しておけばよいでしょう。

これで状態の保存の半分、状態の復帰が実現できました。時計を終了しても、人間が設定を覚えておいて、次に起動するときにオプションをつけて起動すれば、前の状態が再現できます*12。しかし、いちいち人間がこれらの仕事を行うのはばかげています。コンピュータにやらせてしまいましょう。

*12: コマンドラインを編集するためには、[OPT.I] を押しながら実行ファイルのアイコンをダブルクリックします。

すでに述べたように、シェルの終了時に各タスクのコマンドライン文字列は SYSSTOP.SX に保存されます。ということは、ここに前の状態を意味するオプションを書いたコマンドライン文字列が保存されていれば、次に起動されるときには前と同じ状態で起動することになりはしないでしょうか？

たとえば、私たちの時計であれば、12 時制で時刻を表示し、かつ時報を鳴らす状態であった場合、SYSSTOP.SX に記録されるべきコマンドラインを '-tl-jl' としておきます。次に起動されるときには、時計には、この '-tl-jl' が渡されます。この結果、時計は 12 時制の時報ありに設定され、見事に前の状態が復元されたことになります。

それでは、これを実現するコードを書いてみます。スケルトンに追加する位置は BODY.S の 101 行目、

```
cmp.w      # $20,d0      * ウィンドウのセレクト?
beq        WindowSelect  *   ならば WindowSelect へ
```

の直後となります。

まずは、タスクマネージャイベントコードが 31, 「状態の保存」であるかどうかを調べることから始まります。

```
cmp.w      # 31,d0      * 状態の保存?
beq        Save         *   ならば Save へ
```

BODY.S の 117 行目、WindowSelect の rts の直後に Save というラベルを設定し、ここから状態の保存処理を書き始めることにします。

コマンドラインを書き換えるには、各タスクについてタスク名やコマンドライン等の情報を保存しているタスク管理テーブルを利用します。あらかじめ注意しておきますが、ここには重要な情報も記録されているので、必要以上に手を加えてはいけません。

まずは、スタック上に領域を確保してから、自分のタスク管理テーブルを読み込むことにします。

Save:

link	a4, #-512	* 512 バイトの領域を確保
move.w	#-1, -(sp)	* 自分のタスク管理テーブルを取得
pea	-512(a4)	* スタック上の領域に読み込む
SXCALL	\$A35B	* __TSGetTdb
addq.l	# 6, sp	

これで、A4 を終端とするスタック上の領域に、時計自身のタスク管理テーブルがコピーされました。

タスク管理テーブル中のコマンドライン文字列は、オフセット\$5A から LASCII 型で記録されています*13。ここにオプションを書き込みます。前回何が書かれていたかは、とくに意識する必要はありません。コマンドライン文字列の先頭を A0 に収めて、オプションの書き込みに備えます。また、オプションの書き込みはロングワードで行うと便利なので、文字列の先頭にスペースを入れて、偶数アドレスから書き込みが始められるように補正しておきます。

*13:「SX-WINDOW～」初版第1刷、第2刷では ASCII 型と記されていましたが、正しくは LASCII 型です。

lea	-512+\$5a(a4), a0	* A0: 文字列先頭 (文字数)
lea	2(a0), a1	* A1: 文字列実体先頭
move.b	# ' ', 1(a0)	* スペースでアドレス補正

時制の設定を記録します。

move.l	# '-t1', d0	* '-t1'
tst.w	jisei12(a5)	* 12 時制?
bne	Save0	* ならば Save0 へ

move.w	# '0', d0	* '-t0'
Save0:		
move.l	d0, (a1) +	

時報の設定を記録します。

move.l	# '-j1'.shl.8, d0	* '-j1', 0
tst.w	jihou(a5)	* 時報を鳴らす?
bne	Save1	* ならば Save1 へ

```

        move.w    # 'O'.shl.8,d0          *'-j0', 0
Save1:
        move.l    d0,(a1)

```

LASCII 文字列なので、文字数をセットします。この場合、文字数は 7 文字と決まっているので、文字列先頭に 7+1 (補正したスペースの分) を入れます。

```

        move.b    # 7+1,(a0)

```

コマンドライン文字列の設定は完了したので、次は文字列を設定し終わったタスク管理テーブルのコピーを、SX-SYSTEM 内の原本に上書きします。

```

        move.w    # -1,-(sp)              *自分のタスク管理テーブルに書き込む
        pea       -512(a4)               *スタック上のコピーから
        SXCALL    $A35C                  *__TSSetTdb
        addq.l     # 6,sp

```

スタック上に確保した領域を開放して、終了します。

```

        unlk      a4                     *スタック上の領域を開放

        moveq     # 0,d0
        rts

```

このようにして変更された SX-SYSTEM 内のタスク管理テーブルは、このイベントの後、SYSDTOP.SX に、その内容の一部 (つまり、タスク名、コマンドライン文字列等) が記録されます。

以上の処理と、初期化処理がペアとなって、状態の保存が実現できます。

③ 初期化処理ルーチン

初期化ルーチン INIT は、SKELTON.S の中で非常に下位のレベルの初期化を行って、SX アプリケーションとして動作する下地を整えた後で呼び出されます。

前節で挙げた初期化ルーチンの仕事は、

- 1) ワークエリアの初期化
- 2) ウィンドウのオープン
- 3) コントロールのオープン
- 4) 初期画面の描画

の 4 つでしたが、状態の保存によってコマンドラインの解析が必要となったので、

- 5) オプションの解析

が追加されます。

このうち、2) のウィンドウのオープンについては、パラメータの変更程度で実現できます。

- 1) ワークエリアの初期化

いくつかの変数については、すでに初期化の処理が書き込まれていますが、私たちの時計固有の変数の初期化を追加する必要があります。初期化の必要な固有の変数にはどのようなものがあるでしょうか。

- ・ 前回の時刻書き換えシステム時刻 lastTimeUpdate

初期化処理実行時のシステム時刻を調べて代入してもよいのですが、どのみち最初に 1 回は書き換えてもらわなければならないので、はるか以前のシステム時刻、つまり 0 を代入しておきます。こうすることで、次のアイドルイベントでは必ず書き換えが起こります。

このほかの変数については、ウィンドウやコントロールのオープン、オプションの解析などの箇所でも初期化されます。

この初期化の処理は BODY.S の 159 行目

```
st          winAvtive(a5)          * アクティブフラグをセット
```

の直後に置くことにします。

```
clr.l      lastTimeUpdate(a5)      * 前回の書き換え時刻をクリア
```

- 2) ウィンドウのオープン

前述のように、ウィンドウのオープンに関して行わなければならないのは、パラメータの変更程度です。BODY.S では、変更しやすいように、ウィンドウの大きさとウィンドウオプションについてはシンボル化して、ソースの始めのほうに置いてあります。

私たちの時計のウィンドウの大きさは (160, 80)。ウィンドウオプションは、サイズボタンやスクロールボックスなど、付属品はいつさい必要ないので、%0000 ということになります。したがって、BODY.S の 20 行目からは次のように書き換えます。

```
WINOPT  =      %0000      *ウィンドウオプション
WIN_X   =      160      *ウィンドウ初期 x
WIN_Y   =      80      *ウィンドウ初期 y
```

このほかに、ウィンドウタイトルも決めなければいけません。スケルトンでは、winTitle というラベルから始まる LASCII 型の文字列がウィンドウタイトルとなります。私たちの時計のウィンドウタイトルはシンプルに 'Clock' とすることにします。

```
winTitle:
          dc.b      5,'Clock'      *ウィンドウタイトル
```

3) オプションの解析

システムイベントの処理のところで述べたように、状態の保存はシステムイベントでの状態のセーブと、初期化処理での状態の復帰がペアになって実現されます。スケルトンでは '-W' オプションの解析を行っていますが、SX コールを利用しているので、この処理に相乗りすることはできません。別の場所にコマンドラインの解析処理を書くことが必要です。ここでは BODY.S の 144 行目、

```
SXCALL    $A360      *__TGetID
move.l     d0,taskID(a5)      *タスク ID を得る
```

の直前で解析を行うことにします。

コマンドラインのアドレスは変数 cmdLine に収められているので、これを A0 に読み込んで、解析しやすいように ASCIIZ 文字列に変換してからオプション文字の '-' を探します。

```

moveq      #0,d1      *jisei12 に入る値
moveq      #0,d2      *jihou に入る値
move.l     cmdLine(a5),a0      *コマンドラインのアドレス
moveq      #0,d0
move.b     (a0)+,d0      *文字数
sf         (a0,d0.w)      *文字列末尾に$00 を付加
_INIT3:
move.b     (a0)+,d0      *1 文字読み込み
```



```

beq      _INIT4      * $00 なら解析終了
cmp.b    #'-',d0      * オプション文字?
bne      _INIT3      *   でなければ _INIT3 へ

```

オプション文字が³発見できたら、続く文字でオプションの種類を判断し、分岐します。

```

move.b    (a0)+,d0      * もう 1 文字読み込む
beq      _INIT4      * $00 なら解析終了
cmp.b    #'t',d0      * 時制の設定?
beq      _INIT30      *   ならば _INIT30 へ
cmp.b    #'j',d0      * 時報の設定?
beq      _INIT31      *   ならば _INIT31 へ
bra      _INIT3      * それ以外ならば _INIT3 へ

```

それぞれの設定をレジスタ D1, D2 に記録します。

```

_INIT30:      * 時制の設定
              move.b    (a0)+,d1      * さらに 1 文字読み込む
              sub.b     #'O',d1      * 'O'を引くことで$00 か$01 となる
              bra      _INIT3
_INIT31:      * 時報の設定
              move.b    (a0)+,d2
              sub.b     #'O',d2
              bra      _INIT3

```

コマンドラインの末尾まで達したら、それまでにレジスタ D1, D2 に記録された設定を変数に格納します。

```

_INIT4:
              move.w     d1,jisei12(a5)
              move.w     d2,jihou(a5)

```

以上で、コマンドラインの解析は終了し、指定されたオプションの意味にしたがって変数が³設定されました。

4) コントロールのオープン

この処理は BODY.S の 161 行目、

bsr DrawGraph1st * ウィンドウ内部を描画する

の直前に置くことにします。

コントロールのオープン自体は難しいものではありませんが、パラメータが多いため、煩雑に見えるかもしれません。ここでは、チェックボックス 1つを代表として、そのコードを示しておきます。

clr.l	-(sp)	* ユーザワーク
move.w	# 2*16,-(sp)	* チェックボックス
move.w	# 1,-(sp)	* max
move.w	# 0,-(sp)	* min
move.w	jihou(a5),-(sp)	* 初期値
move.w	# -1,-(sp)	* 可視
pea	chkBoxTitle(pc)	* タイトル (が、表示しない)
pea	chkBoxRect(pc)	* チェックボックスのレクタングル
pea	winPtr(a5)	* ウィンドウレコード
SXCALL	\$A289	* __CMOpen
lea	26(sp),sp	
move.l	a0,chkBoxHdl(a5)	* コントロールレコードへのハンドルを保存

チェックボックスの初期値として、オプションの指定を格納した jihou を指定していることに注目してください。タイトルとして chkBoxTitle を指定してはいますが、実際には表示されることはありません。なぜならば、その次の chkBoxRect にチェックボックスそのものの大きさしか持たせず、タイトルはこの中に含まれないからです。タイトルを含むようにレクタングルを設定すると、タイトルは CMDraw によって表示されるようになりますが、タイトルの文字をクリックしても、チェックボックスを押したことになるという不都合が起こります。

このほかのコントロールでは、このような注意は必要ありません。

5) 初期画面の描画

スケルトンでは、初期画面を描画するために DrawGraph1st というサブルーチンを呼んでいます。このサブルーチンは、リージョンやスクリプトを作成する必要がある場合を考えて用意しましたが、私たちの時計ではとくにそういった必要はありません。初期画面の描画といっても、通常の時刻や、そのほかの部分の描画となんら変わりはないのです。

そこで、ウィンドウ内部の初期状態を描画させるために、DrawGraph1st を利用する必要はまったくないので、例の 2 つの画面描画ルーチンと置き換えてしまいます。すなわち、BODY.S の 161 行目、

```
bsr      DrawGraphlst      *ウィンドウ内部を描画する
```

のかわりに

```
bsr      DrawTime
bsr      DrawOther
```

を置くことにします。

この結果、169 行目の

```
DrawGraphlst:      *ウィンドウ内部の描画の
                   *準備をするサブルーチン
                   * (何もしない)

rts
```

は不要となりますから、削除してかまいません。

④ 終了処理ルーチン

終了処理ルーチンは、初期化ルーチン INIT、あるいは各イベントの処理ルーチンで何か異常な事態、あるいは終了しなければならない事態が発生したとき、メインループから実行に移されます。

ここで行う処理は、コントロールやウィンドウのクローズですが、ウィンドウのクローズはすでに用意されているので、書かなければならないのはコントロールのクローズだけです。BODY.S178 行目の

```
_TINI:      * [終了処理]
```

の直後から書き始めることにします。

コントロールのクローズは、あるウィンドウ中にあるコントロール (複数) を 1 個 1 個クローズしてもよいのですが、終了処理のように、一度にすべてをクローズしたい場合は便利な方法があります。SX コール \$A28B CMKill がそれで、クローズしたいコントロールの乗ったウィンドウを指定するだけで、すべてのコントロールをクローズしてくれます。

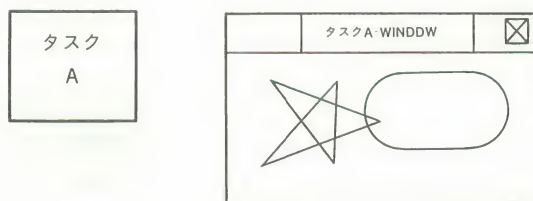
```
pea      winPtr(a5)
SXCALL   $A28B      * _CMKill
addq.l   #4,sp
```

COLUMN 前回の状態の再現の仕組み

「前回の状態の再現」の仕組みは少々わかりにくいかもしれないので、ここで解説しておくことにします。

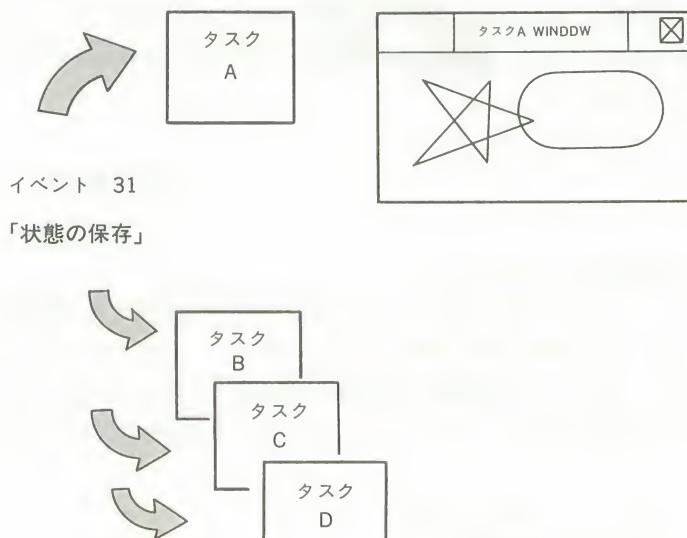
例として、SX シェル上で A というアプリケーションが動作しているところを考えてみましょう。A はウィンドウを持っていて、その中に何か絵を表示していたとします (図 a)。

■図 a



突然、シェルは終了しなければならなくなりました。ユーザがシステムアイコンから「終了」か、「再起動」を選択したのか、それとも電源スイッチが切られて、いままさに電源切断の予告段階に入ったところなのか、それはわかりませんが、とにかく終了を迫られています。このとき、シェルは、そのとき動作しているすべてのタスクに対して、システムイベント(タスクマネージャイベントコード 31「状態の保存」)を発生し、状態の保存をしよう呼びか

■図 b



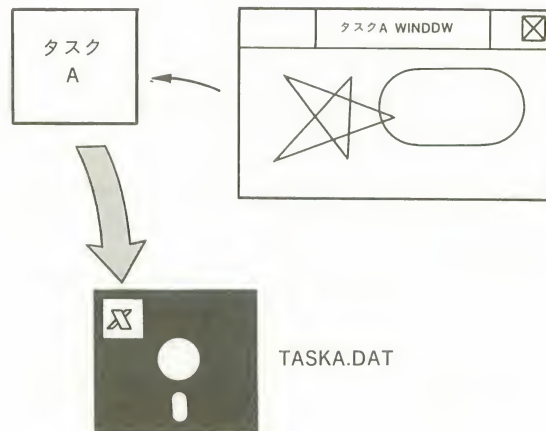
けます (図 b)。

A にもシステムイベントが通知され、A は状態の保存にとりかかります。A が、この次に起動されたときに、現在の状態を再現するためには、次の 3 つの情報が必要です。

- a) 現在表示している絵のデータ
- b) 現在のウィンドウの位置
- c) 現在のウィンドウの大きさ

これらの情報さえあれば、次に起動されたときに、現在のウィンドウの位置に、現在のウィンドウの大きさで、現在表示している絵を表示することができるわけです。ユーザにとって、それは現在とまったく同じ状態に見えるはずで。このうち、A は a) の情報だけファイルのかたちで保存し、b) と c) は保存しません。これがなぜかはすぐにわかります (図 c)。

■図 c

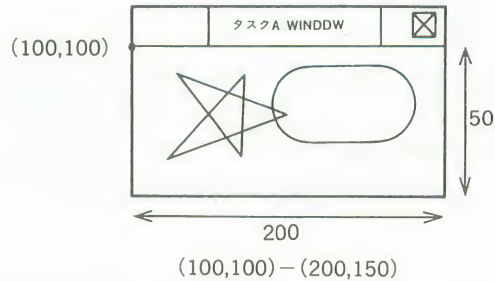


すべてのタスクに「状態の保存」を通知し終わると、シェルはいよいよ本格的に終了を始めます。シェル自身の終了の前に、すべてのタスクのタスク管理テーブルの一部+αをファイルに保存します。ファイル名は SYSDTOP.SX。SXWIN.X と同じディレクトリに毎回作成されているこのファイルには、こんな意味があったのです。ここに保存される各タスクに関するおもな情報としては、次のようなものがあります。

- ア) タスク名 (原則としてフルパスのファイル名)
- イ) 起動されたときに指定されたコマンドライン文字列
- ウ) タスクの走行状態や起動モード
- エ) そのタスクのウィンドウの位置や大きさ (複数のウィンドウを持っている場合は、メインのウィンドウのみ。ウィンドウを持たない場合、ここは空白)

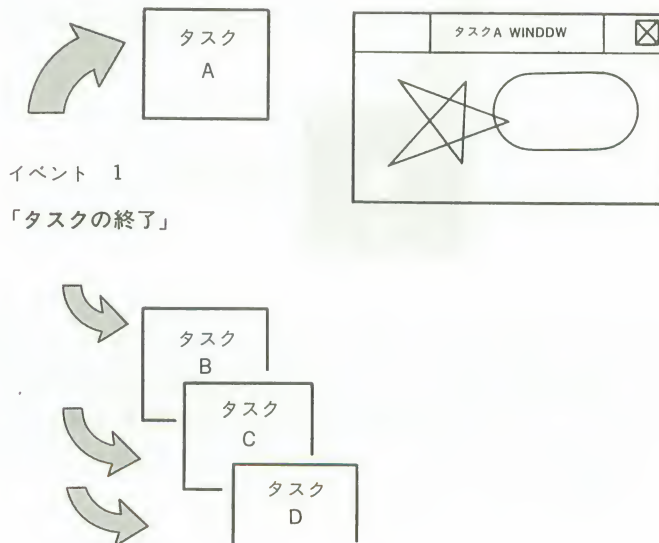
エ)としてウィンドウの位置や大きさを保存してくれるので、結局、A自身は自分のウィンドウの位置や大きさを保存する必要がなかったわけです。シェルはAについて、そのタスク名やコマンドライン文字列などといっしょに、その時点でのAのウィンドウの位置や大きさを記録しました。このとき、Aのウィンドウのウィンドウコンテンツは、グローバル座標系で(100, 100)-(200, 150)だったとしましょう(図d)。

■図d



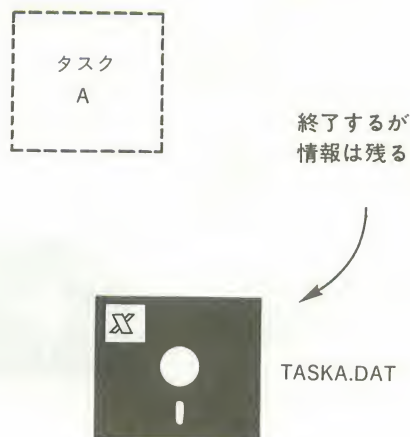
続いて、シェルはすべてのタスクにシステムイベント、今度はタスクマネージャイベントコード1「タスクの終了」を通知し、それぞれの終了処理を行うよう要求します(図e)。

■図e



「タスクの終了」要求は、Aにも届き、Aは自分のウィンドウをクローズしたり、作成したメモリブロックを開放したりといった終了処理を行い、最終的にタスクを終了させます。このとき、タスクの状態についての情報がファイルのかたちで残されているのがミソです(図f)。

■図 f



すべてのタスクに「タスクの終了」を要請し終わると、シェルは今度こそ自分の終了処理を始め、最終的に自分自身を終了させ、制御は Human に戻ります。

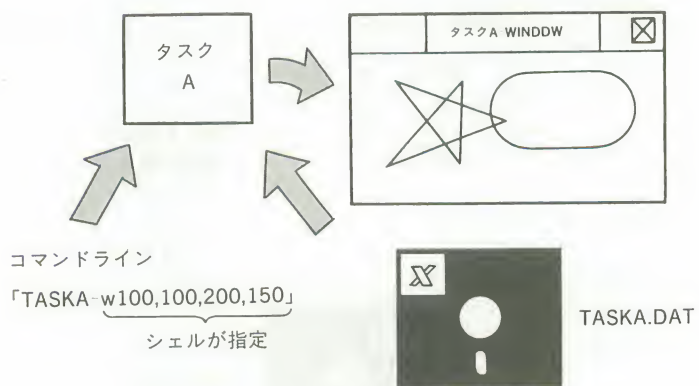
さて、しばらくして、ふたたび SX シェルが起動されることになりました。シェルは各種マネージャの初期化など、自分自身のための初期化処理を行ってから、前回の状態の再現にとりかかります。SYSDTOP.SX を読み出して、そこに登録されているタスクを次々にファイルなどから起動します。このとき、各タスクには SYSDTOP.SX に保存されていたコマンドライン文字列が与えられるわけですが、シェルはここでちょっとした細工を行います。SYSDTOP.SX に保存されていた、各タスクのウィンドウの位置と大きさ、これをあたかもコマンドライン上で -W オプションが指定されたかのように、コマンドライン文字列につけ加えるのです。A に関していえば、-W100, 100, 200, 150 という文字列がつけ加えられることになります。

ふたたび起動された A は、コマンドライン文字列の中に -W100, 100, 200, 150 という指定があることに気づきます。-W オプションが指定されていた場合、アプリケーションはそれにしたがってウィンドウを開かなければならないので、A はこの位置にウィンドウを開きます。そして、ファイルとして、かつて保存した情報が残されていることを発見すると、その情報にしたがってウィンドウ内部に絵を描きます (図 g)。

このようにして、ふたたび起動された A は、結果的に前回の状態を再現することができたことになります。いくつも走行していたタスクが、それぞれこのような処理を行うことによって、SX シェル終了直前の状態が完全に再現されるのです。

なお、以上のような処理は、システムアイコンの「画面状態の保存」設定で、「常時保存する」を選択した場合に行われます。「常時保存しない」を選択している場合、同じダイアログ内の「現在の状態を保存」ボタンを押したときに、全タスクに「状態の保存」要請が発せられ、シェルの終了時には、いきなり「タスクの終了」要請が通知されることになります。

■図 g



1³ モジュールの作成とリンク

プログラムを機能単位で分割して、修正やほかのプログラムへの流用を容易にする手法を「モジュール化」と呼びます。プログラムのソースの分割は、モジュール化を行ううえでたいへん基本的、かつ重要な手法の1つです。モジュールという独立した機能単位に分割することによって、プログラマが一度に見渡すべき範囲は小さくて済みます。巨大なソースの中に埋もれてしまうと発見しにくくなるバグも、こうして比較的小さなモジュールに区切ることによって、発見が容易となります。さらに、こうすることによって、プログラムのドキュメント性も向上します*1。

*1: 長々と続く文章よりも、的確に句読点を打ち、段落分け、章分けを行った文章のほうが読みやすいのと同じことです。

モジュールは、プログラムとしてある程度独立しているため、モジュールごとのデバッグが可能となることも利点の1つです*2。バグが発見できた場合、再コンパイル/アセンブルする必要があるのは、分割された小さなソースが1つですから、実行ファイルが生成されるまでの時間も短縮できます。

*2: たとえば、ごく小さなテスト用のメインモジュールを用意して、そこで最小限の初期化と目的のモジュールの呼び出しを行えば、そのモジュールが担う機能だけについて動作チェックが可能となります。

難点として、ファイルが多くなってくると管理が難しくなってくることが挙げられますが、これには make と呼ばれるツールを使って対処することができます。make は C コンパイラ Ver.2.0 に付属しますが、フリーソフトウェアの make も入手可能ですので、興味のある方はぜひ利用してみてください(87 ページのコラム参照)。本書では make に関して詳しい説明は省きますが、各サンプルプログラムを生成するための makefile を示しておくことにします。

本書で示すスケルトンのコードは、SKELTON.S と BODY.S に分割されています。前者は、「メインモジュール」としてもっとも基本的な、プログラムの根幹をなす部分が収められており、後者は、ここから呼び出されるさまざまなサブルーチンが収められている「初期化&終了&イベント処理モジュール」です。たった2つの分割ではありますが、私たちの時計を作成するにあたり、SKELTON.S にはまったく手を触れずにすんだことから、ファイル分割、およびモジュール化の意義をいくらか汲み取っていただけたと思います。

さらに大規模なアプリケーションを作成する場合には、BODY.S をさらに分割することが考えられます。アプリケーションの規模や、今後のアプリケーションへの流用の可能性なども考えて、適当に分割して利用するのもよいでしょう。

いまの規模でも、分割によってメリットが得られると考えられるのは描画ルーチンです。単純にウィンドウを出すだけの、スケルトンにわずかに手を加えたテストプログラムと、描画ルーチンのモジュールを組み合わせることによって、描画ルーチンによる表示のぐあいを確かめる

ことができます。文字の表示位置などの微調整を行うときに有効でしょう。

前節で作成したコードをソースファイルとして完成させたものが、MyClock.S (リスト 2) と、変数の定義を行っている WORK.INC (リスト 1) です。MyClock.S はスケルトンの BODY.S に、WORK.INC はスケルトンの同名のファイルに手を加えたもので、SXCALL.MAC, SKELTON.S とリンクして使用します。

■リスト 1 WORK.INC

```

1 *
2 *           SX-WINDOW
3 *           サンプルプログラム # 1
4 *
5 *           ワーク定義用インクルードファイル
6 *
7
8 STKSIZE      =      2*1024           * スタックサイズ
9
10 *           ワークの内容の定義
11
12             .offset 0
13 cmdLine:     * コマンドラインのアドレス
14             ds.l    1               * を保存するワーク
15 envPtr:      * 環境のアドレスを
16             ds.l    1               * 保存するワーク
17 winRect:     * ウィンドウレクタングル
18             ds.l    2               * レコード
19 paramFlg:    * コマンドラインの
20             ds.w    1               * 解析結果を示す
21 eventRec:    * フラグ
22 eventRec_what: * イベントレコードの先頭
23             ds.w    1               * イベントコード
24 eventRec_whom1: * 第1引数
25             ds.l    1
26 eventRec_when: * イベント発生時
27             ds.l    1
28 eventRec_whom2: * 第2引数
29             ds.l    1
30 eventRec_what2:
31             ds.w    1               * タスクマネージャイ
32             * ベントの種類
33 eventRec_taskID:
34             ds.w    1               * 送り手のタスクID
35             * イベントマスクを保存する
36             ds.w    1               * ワーク
37 taskID:      * タスクIDを
38             ds.l    1               * 保存するワーク
39 winPtr:      * ウィンドウレコードを
40             ds.b    $72             * 作成する場所
41 winActive:   * アクティブフラグ
42             ds.w    1
43
44 timeStr:     *
45             ds.b    9               * 時刻文字列+1 ($00)
46             .even

```

```

47 jiseil2:
48             ds.w    1           * 12時制フラグ
49 jihou:
50             ds.w    1           * 時報フラグ
51
52 lastTimeUpdate:
53             ds.l    1           * 最後に時刻を更新した時刻
54
55 chkBoxHdl:
56             ds.l    1           * チェックボックスの
                                   * コントロールレコードへの
                                   * ハンドル
57 rad1Hdl:
58             ds.l    1           * ラジオボタン1の
                                   * コントロールレコード
                                   * へのハンドル
59 rad2Hdl:
60             ds.l    1           * ラジオボタン2の
                                   * コントロールレコードへの
                                   * ハンドル
61
62 WORKSIZE:
63             * ワークの終了

```

■リスト2 MyClock.S

```

1  *
2  *          SX-WINDOW
3  *          MyClock.s
4  *
5  *          初期化&終了&イベント処理モジュール
6  *
7
8          .include      DOSCALL.MAC
9          .include      SXCALL.MAC
10
11         .xdef      _INIT, _TINI
12         .xdef      IDLE
13         .xdef      MSLDOWN, MSLUP, MSRDOWN, MSRUP
14         .xdef      KEYDOWN, KEYUP
15         .xdef      UPDATE, ACTIVATE
16         .xdef      SYSTEM1, SYSTEM2, SYSTEM3, SYSTEM4
17
18         .include      WORK.INC           * ワークエリアの内容
                                           * を定義するファイル
19
20 WINOPT      =      %0000           * ウィンドウオプション
21 WIN_X       =      160             * ウィンドウ初期x
22 WIN_Y       =      80             * ウィンドウ初期y
23
24         .text
25 MSLUP:
26 MSRDOWN:
27 MSRUP:
28 KEYDOWN:
29 KEYUP:
30 SYSTEM3:
31 SYSTEM4:
32         moveq     #0, d0           * [ レフトアップイベント ]
                                           * [ ライトダウンイベント ]
                                           * [ ライトアップイベント ]
                                           * [ キーダウンイベント ]
                                           * [ キーアップイベント ]
                                           * [ システムイベント3 ]
                                           * [ システムイベント4 ]
                                           * 以上のイベントでは
                                           * なにもしない
33
34         rts
35 IDLE:
36         move.l     eventRec_when(a5), d1
                                           * [ アイドルイベント ]
                                           * イベント発生時刻を
                                           * D1に

```

```

37      move.l    d1, d0      * 前回の書き換え
38      sub.l     lastTimeUpdate(a5), d0 * システム時刻からの
39      cmp.l     #100, d0    * 経過時間を求める
40      bcs       IDLE9      * 1秒以上経過?
41                                     * でなければIDLE9へ
42      move.l    d1, lastTimeUpdate(a5) * イベント発生時刻を保存
43      bsr       DrawTime
44
45      tst.w     jihou(a5)    * 時報を鳴らす?
46      beq       IDLE9      * 鳴らさないならIDLE9へ
47
48      moveq     #$56, d0    * _TIMEGET
49      trap      #15         * _TICS呼び出し
50      tst.w     d0          * 00分00秒?
51      bne       IDLE9      * でなければIDLE9へ
52
53      move.w     #2, -(sp)   * 1回
54      SXCALL    $A2D7       * __DMBeep
55      addq.l    #2, sp
56 IDLE9:
57      moveq     #0, d0
58      rts
59                                     * of IDLE
60
61 MSLDOWN:
62                                     * [ レフトダウンイベント ]
63      move.l    eventRec_whoml(a5), a0
64      lea       winPtr(a5), a2
65      cmp.l     a2, a0
66      bne       MSLD9
67
68      tst.b     winActive(a5) * 現在ウィンドウは
69                                     * アクティブか?
70      bne       MSLD1       * アクティブならMSLD1へ
71
72      pea       (a2)
73      SXCALL    $A1FE       * __WMSelect
74      addq.l    #4, sp
75      bra       MSLD9      * アクティブにするだけ
76
77 MSLD1:
78      pea       eventRec(a5)
79      pea       (a2)
80                                     * ウィンドウ処理
81      SXCALL    $A3A2       * __SXCallWindM
82      addq.l    #8, sp
83      tst.l     d0
84                                     * どこも操作されなかった?
85      beq       MSLD9      * ならばMSLD9へ
86
87      cmp.w     #7, d0
88      beq       CloseBttn  * クローズボタン?
89                                     * ならばCloseBttnへ
90
91      cmp.w     #3, d0
92      bne       MSLD9      * ウィンドウコンテンツ?
93                                     * でなければMSLD9へ
94
95      clr.l     -(sp)
96      clr.l     -(sp)
97      clr.l     -(sp)
98      pea       eventRec(a5)
99                                     * dRectPtr (省略)
100                                     * ctrlHdlH (省略)
101                                     * ctrlHdlV (省略)
102                                     * イベントレコード
103                                     * へのポインタ
104      pea       (a2)
105                                     * ウィンドウレコード
106                                     * へのポインタ
107      SXCALL    $A3A3
108      lea       20(sp), sp
109      tst.l     d0
110      beq       MSLD9      * コントロール上?
111                                     * でなければMSLD9へ

```



```

96
97      cmp.l   chkBoxHdl (a5), a0      * チェックボックスが
                                         * 操作された?
98      beq     MSLD_ChkBox             *   ならばMSLD_ChkBoxへ
99      cmp.l   rad1Hdl (a5), a0      * ラジオボタン1が
                                         * 操作された?
100     beq     MSLD_Rad1               *   ならばMSLD_Rad1へ
101     cmp.l   rad2Hdl (a5), a0      * ラジオボタン2が
                                         * 操作された?
102     beq     MSLD_Rad2               *   ならばMSLD_Rad2へ
103     bra     MSLD9                   *   これら以外の場合は
                                         *   MSLD9へ

104
105 MSLD_ChkBox:
106     move.l   chkBoxHdl (a5), -(sp)   * チェックボックスへの
                                         * ハンドル
107     SXCALL   $A291                  *   __CMValueGet
108     addq.l   #4, sp
109     move.w   d0, jihou (a5)
110     bra     MSLD9                   *   MSLD9へ

111 MSLD_Rad1:
112     moveq    #1, d1                 *   ラジオボタン1をonに
113     moveq    #0, d2                 *   ラジオボタン2をoffに
114     bra     MSLD_Rad
115 MSLD_Rad2:
116     moveq    #0, d1                 *   ラジオボタン1をoffに
117     moveq    #1, d2                 *   ラジオボタン2をonに
118 MSLD_Rad:
119     move.w   d1, jisei12 (a5)        *   jisei12に値を設定
120
121     move.w   d1, -(sp)
122     move.l   rad1Hdl (a5), -(sp)    *   ラジオボタン1に値を設定
123     SXCALL   $A290                  *   __CMValueSet
124     addq.l   #6, sp
125     move.w   d2, -(sp)
126     move.l   rad2Hdl (a5), -(sp)    *   ラジオボタン2に値を設定
127     SXCALL   $A290                  *   __CMValueSet
128     addq.l   #6, sp
129     bsr     DrawTime                *   切り替えられた時刻
                                         *   で時刻を再描画

130 *
131 MSLD9:
132     moveq    #0, d0
133     rts
134
135 CloseBttn:
136     moveq    #-1, d0
137     rts
138
139 UPDATE:
140     pea     winPtr (a5)
141     SXCALL   $A20D                  *   [ アップデートイベント ]
142     addq.l   #4, sp                 *   __WMUpdate
143                                         *   アップデート開始
144     bsr     DrawTime
145     bsr     DrawOther
146
147     pea     winPtr (a5)
148     SXCALL   $A20E                  *   __WMUpdtOver
149     addq.l   #4, sp                 *   アップデート終了
150
151     moveq    #0, d0
152     rts
153

```

```

154 ACTIVATE:                                     * [ アクティベイトイベント ]
155         move.l  eventRec_whom1(a5), d0
156         beq     ACT9
157         lea     winPtr(a5), a0
158         cmp.l   a0, d0
159         bne     ACT0
160         st      winActive(a5)
161         bra     ACT9
162 ACT0:
163         sf      winActive(a5)
164 ACT9:
165         moveq   #0, d0
166         rts
167
168 SYSTEM1:                                     * [ システムイベント1 ]
169 SYSTEM2:                                     * [ システムイベント2 ]
170         move.w  eventRec_what2(a5), d0
171         cmp.w   #1, d0
172         beq     AllClose
173         cmp.w   #2, d0
174         beq     AllClose
175         cmp.w   #$20, d0
176         beq     WindowSelect
177         cmp.w   #31, d0
178         beq     Save
179
180         moveq   #0, d0
181         rts
182
183 AllClose:
184         moveq   #-1, d0
185         rts
186
187 WindowSelect:
188         pea     winPtr(a5)
189         SXCALL  $A1FE
190         addq.l  #4, sp
191
192         moveq   #0, d0
193         rts
194
195 Save:
196         link    a4, #-512
197
198         move.w  #-1, -(sp)
199         pea     -512(a4)
200         SXCALL  $A358
201         addq.l  #6, sp
202
203         lea     -512+$5a(a4), a0
204         lea     2(a0), a1
205         move.b  #' ', 1(a0)
206         move.l  #'-t1', d0
207         tst.w   jisei12(a5)
208         bne     Save0
209
210         move.w  #'0 ', d0
211 Save0:
212         move.l  d0, (a1)+
213

```

```

214      move.l    #'-j1', sh1.8, d0      * '-j1', 0
215      tst.w     jihou(a5)               * 時報を鳴らす?
216      bne       Save1                   *   ならばSave1へ
217
218      move.w     #'0', sh1.8, d0         * '-j0', 0
219 Save1:
220      move.l     d0, (a1)
221
222      move.b     #7+1, (a0)
223      move.w     #-1, -(sp)              * 自分のタスク管理テーブル
                                           *   に書き込む
                                           *   スタック上のコピーから
                                           *   __TSSetTdb
224      pea       -512(a4)
225      SXCALL     $A35C
226      addq.l     #6, sp
227
228      unlk       a4                      * スタック上の領域を開放
229
230      moveq      #0, d0
231      rts
232
233 _INIT:
                                           * [ アプリケーション
                                           *   の初期化を行なう ]
234      move.l     winRect(a5), d0
235      move.w     paramFlg(a5), d1
236      btst      #0, d1
                                           * '-W' オプションが
                                           *   指定された?
                                           *   指定されていないければ
                                           *   _INIT0へ
237      beq       _INIT0
238
239      move.l     winRect+4(a5), d1
240      beq       _INIT1
                                           * 正しいレクタングルが
                                           *   指定されたかどうか
                                           *   を調べる
241      tst.w     d1
242      cmp.w     d0, d1
243      ble       _INIT1
244      swap      d0
245      swap      d1
246      cmp.w     d0, d1
247      bgt       _INIT2
248      swap      d0
249      swap      d1
250      bra       _INIT1
251 _INIT0:
252      SXCALL     $A35E                    * __TSGetWindowPos
253      move.l     d0, winRect(a5)         * デフォルト位置を得る
254 _INIT1:
255      add.l      #WIN_X*$10000+WIN_Y, d0 * ウィンドウレクタングル
                                           *   を作成
256      move.l     d0, winRect+4(a5)
257 _INIT2:
258      moveq      #0, d1                    * jisei12に入る値
259      moveq      #0, d2                    * jihouに入る値
260      move.l     cmdLine(a5), a0          * コマンドラインのアドレス
261      moveq      #0, d0
262      move.b     (a0)+, d0                 * 文字数
263      sf         (a0, d0.w)               * 文字列末尾に$00を付加
264 _INIT3:
265      move.b     (a0)+, d0                 * 1文字読み込み
266      beq        _INIT4                   * $00なら解析終了
267      cmp.b     #'-', d0                  * オプション文字?
268      bne        _INIT3                   *   でなければ_INIT3へ
269
270      move.b     (a0)+, d0                 * もう一文字読み込む
271      beq        _INIT4                   *   $00なら解析終了

```

272	cmp. b	#' t', d0	* 時制の設定?
273	beq	_INIT30	* ならば_INIT30へ
274	cmp. b	#' j', d0	* 時報の設定?
275	beq	_INIT31	* ならば_INIT31へ
276	bra	_INIT3	* それ以外ならば_INIT3へ
277	_INIT30:		* 時制の設定
278	move. b	(a0) +, d1	* さらに一文字読み込む
279	sub. b	#' 0', d1	* '0' を引くことで
			* \$00か\$01となる
280			
281	_INIT31:		* 時報の設定
282	move. b	(a0) +, d2	
283	sub. b	#' 0', d2	
284	bra	_INIT3	
285	_INIT4:		
286	move. w	d1, jisei12 (a5)	
287	move. w	d2, jihou (a5)	
288			
289	SXCALL	\$A360	* _TSGetID
290	move. l	d0, taskID (a5)	* タスクIDを得る
291			
292	move. l	d0, -(sp)	* タスクID
293	move. w	#-1, -(sp)	* クローズボタンあり
294	move. l	#-1, -(sp)	* もっとも手前に
295	move. w	#\$20*16+WINOPT, -(sp)	* 標準ウィンドウ
296	move. w	#-1, -(sp)	* 可視
297	pea. l	winTitle (pc)	* ウィンドウタイトル
298	pea. l	winRect (a5)	* ウィンドウレクタングル
299	pea	winPtr (a5)	* ワーク上に作成
300	SXCALL	\$A1F9	* _WMOpen
301	lea. l	26 (sp), sp	* ウィンドウを開く
302	tst. l	d0	* エラー?
303	bmi	_INIT_Err	* ならば_INIT_Errへ
304	st	winActive (a5)	* アクティブフラグをセット
305	clr. l	lastTimeUpdate (a5)	* 前回の書き換え
			* 時刻をクリア
306			
307	clr. l	-(sp)	* ユーザーワーク
308	move. w	#\$2*16, -(sp)	* チェックボックス
309	move. w	#1, -(sp)	* max
310	move. w	#0, -(sp)	* min
311	move. w	jihou (a5), -(sp)	* 初期値
312	move. w	#-1, -(sp)	* 可視
313	pea	chkBoxTitle (pc)	* タイトル (が表示しない)
314	pea	chkBoxRect (pc)	* チェックボックス
			* のレクタングル
315	pea	winPtr (a5)	* ウィンドウレコード
316	SXCALL	\$A289	* _CMOpen
317	lea	26 (sp), sp	
318	move. l	a0, chkBoxHdl (a5)	* コントロールレコードへの
			* ハンドルを保存
319			
320	move. w	jisei12 (a1), d1	
321			
322	clr. l	-(sp)	* ユーザーワーク
323	move. w	#\$1*16, -(sp)	* ラジオボタン
324	move. w	#1, -(sp)	* max
325	move. w	#0, -(sp)	* min
326	move. w	d1, -(sp)	* 初期値 (jisei12と同じ)
327	move. w	#-1, -(sp)	* 可視
328	pea	radlTitle (pc)	* タイトル (が表示しない)
329	pea	radlRect (pc)	* ラジオボタン1の
			* レクタングル
330	pea	winPtr (a5)	* ウィンドウレコード


```

331      SXCALL    $A289
332      lea       26(sp), sp
333      move.l    a0, rad1Hdl (a5)
334
335      eor.w     #1, d1
336
337      clr.l     -(sp)
338      move.w    #1<16, -(sp)
339      move.w    #1, -(sp)
340      move.w    #0, -(sp)
341      move.w    d1, -(sp)
342      move.w    #-1, -(sp)
343      pea       rad2Title(pc)
344      pea       rad2Rect(pc)
345
346      pea       winPtr(a5)
347      SXCALL    $A289
348      lea       26(sp), sp
349      move.l    a0, rad2Hdl (a5)
350
351      bsr       DrawTime
352      bsr       DrawOther
353
354      moveq     #0, d0
355      rts
356      _INIT_Err:
357      moveq     #-1, d0
358      rts
359      DrawTime:
360
361      pea       winPtr(a5)
362      SXCALL    $A131
363      addq.l    #4, sp
364
365      moveq     #$56, d0
366      trap      #15
367      move.l    d0, d1
368      moveq     #$57, d0
369      trap      #15
370
371      move.w    #' ', d7
372
373      swap      d0
374
375      tst.w     jisei12(a5)
376      beq       DrawTime2
377
378      cmp.w     #12, d0
379      bcc       DrawTime0
380
381      move.w    #' AM', d7
382      bra       DrawTime1
383
384      move.w    #' PM', d7
385      sub.w     #12, d0
386
387      tst.w     d0
388      bne       DrawTime2

```

* __CMOpen
 * コントロールレコードへの
 * ハンドルを保存
 * jisei12(0->1, 1->0)
 * ユーザーワーク
 * ラジオボタン
 * max
 * min
 * 初期値 (jisei12の逆)
 * 可視
 * タイトル (が表示しない)
 * ラジオボタン2の
 * レクタングル
 * ウィンドウレコード
 * __CMOpen
 * コントロールレコードへの
 * ハンドルを保存
 * 時刻を描画する
 * サブルーチン
 * GMSetGraph
 * _TIMEGET
 * TOCS呼び出し
 * _TIMEBIN
 * TOCS呼び出し
 * 12時制の場合、後で
 * AM/PMの文字列を入れる
 * 上位ワードの時の位
 * を下位に
 * 12時制かどうかのフラグ
 * 24時制ならDrawTime2へ
 * 12時以降?
 * ならば午後なので
 * DrawTime0へ
 * 午前
 * 12時間引く
 * 0時?
 * でなければDrawTime2へ

```

388      move.w  #12, d0      * 12時に直す
389 DrawTime2:
390      swap    d0           * 上位/下位ワードを
                          * 元の順に戻す
391      move.l  d0, d1      * D1 : 時刻
392      lea     timeStr(a5), a1 * A1 : 文字列が返るバッファ
                          * へのポインタ
393      moveq   #$5b, d0    * TIMEASC
394      trap    #15         * TOCS呼び出し
395
396      move.w  #$100, -(sp) * バックグラウンド
                          * カラーで描画
397      SXCALL  $A144       * GMPenMode
398      addq.l  #2, sp
399      pea     timeRectInside(pc) * 塗り潰すべき四角形の内側
400      SXCALL  $A173       * GMFillRect
401      addq.l  #4, sp
402
403      move.w  #2, -(sp)    * 24×24
404      SXCALL  $A18B       * GMFontKind
405      addq.l  #2, sp
406      move.w  #%00011, -(sp) * イタリック+ボールド
407      SXCALL  $A18C       * GMFontFace
408      addq.l  #2, sp
409      move.w  #0, -(sp)    * pset
410      SXCALL  $A18D       * GMFontMode
411      addq.l  #2, sp
412      move.w  #11, -(sp)   * 黒
413      SXCALL  $A147       * GMForeColor
414      addq.l  #2, sp
415
416      move.l  #$0018_0008, -(sp) * (24, 8)
417      SXCALL  $A16E       * GMMove
418      addq.l  #4, sp
419      pea     timeStr(a5)   * 時刻文字列のバッファ
420      SXCALL  $A192       * GMDrawStrZ
421      addq.l  #4, sp
422
423      move.w  #0, -(sp)    * 12×12
424      SXCALL  $A18B       * GMFontKind
425      addq.l  #2, sp
426      move.w  #%00001, -(sp) * ボールド
427      SXCALL  $A18C       * GMFontFace
428      addq.l  #2, sp
429
430      move.l  #$0088_0014, -(sp) * (136, 20)
431      SXCALL  $A16E       * GMMove
432      addq.l  #4, sp
433
434      swap    d7
435      clr.w   d7           * これでD7は'AM', 0, 0
                          * という形式になる
436      move.l  d7, -(sp)    * スタックフレーム上に
                          * ASCII文字列として置く
437      pea     (sp)        * スタックフレーム上の
                          * 文字列を指す
438      SXCALL  $A192       * GMDrawStrZ
439      addq.l  #8, sp
440
441      rts                * of DrawTime
442
443 DrawOther:
                          * 時刻以外を描画する
                          * サブルーチン
444      pea     winPtr(a5)

```

```

445      SXCALL $A131      # GMSetGraph
446      addq.l #4, sp
447
448      pea    timeRectOutside(pc)
449      SXCALL $A1A2      # GMShadowRect
450      addq.l #4, sp
451
452      pea    winPtr(a5)
453      SXCALL $A28E      # CMDraw
454      addq.l #4, sp
455
456      move.w #0, -(sp)   # 12×12
457      SXCALL $A18B      # GMFontKind
458      addq.l #2, sp
459      move.w #%00000, -(sp) # 装飾なし
460      SXCALL $A18C      # GMFontFace
461      addq.l #2, sp
462
463      move.l #$0054_0040, -(sp) # (84, 64)
464      pea    chkBoxTitle(pc) # チェックボックスの
                                # タイトル文字列(ASCII2)
                                # GMShadowStrZ
465      SXCALL $A1A1
466      addq.l #8, sp
467
468      move.l #$000C_0034, -(sp) # (12, 52)
469      pea    radGTitle(pc) # ラジオボタングループ
                                # タイトル文字列(ASCII2)
                                # GMShadowStrZ
470      SXCALL $A1A1
471      addq.l #8, sp
472
473      move.l #$003C_0028, -(sp) # (60, 40)
474      pea    rad1Title(pc) # ラジオボタン1の
                                # タイトル文字列(ASCII2)
                                # GMShadowStrZ
475      SXCALL $A1A1
476      addq.l #8, sp
477
478      move.l #$0078_0028, -(sp) # (120, 40)
479      pea    rad2Title(pc) # ラジオボタン2の
                                # タイトル文字列(ASCII2)
                                # GMShadowStrZ
480      SXCALL $A1A1
481      addq.l #8, sp
482
483      rts      # of DrawOther
484
485
486 _INI:      # [ 終了処理 ]
487      pea    winPtr(a5)
488      SXCALL $A28B      # __CMKill
489      addq.l #4, sp
490
491      pea    winPtr(a5)
492      SXCALL $A1FB      # ウィンドウをクローズする
493      addq.l #4, sp      # __WMClose
494                        # WMDisposeでないことに注意
495      moveq #0, d0
496      rts
497
498      .even      # [ 固定データ ]
499 winTitle:
500      dc.b     5, 'Clock'      # ウィンドウタイトル
501
502 chkBoxTitle:
503      dc.b     '時報を鳴らす', 0
504 radGTitle:

```

```

505          dc. b    ' 時制', 0
506 rad1Title:      dc. b    ' 12時制', 0
507          dc. b    ' 24時制', 0
508 rad2Title:      dc. b    ' 24時制', 0
509          dc. b    ' 24時制', 0
510          . even
511 timeRectOutside: dc. w    4, 4, 156, 36
512 timeRectInside: dc. w    4+2, 4+2, 156-2, 36-2
513 chkBoxRect:     dc. w    60, 64-6, 78, 82
514 rad1Rect:       dc. w    60, 52+2, 96, 62+2
515 rad2Rect:       dc. w    120, 52+2, 156, 62+2
516          dc. w
517          . end
518          . end
519          . end
520          . end
521          . end
522          . end
523          . end
524          . end

```

先ほど組み立てたコードが、どのように BODY.S の中に埋めこまれているか、また、必要な変数が WORK.INC の中でどのように定義されているかを確認してください。

アセンブルを開始する前に、準備として、SXCALL.MAC, WORK.INC をインクルード可能なディレクトリに、SKELTON.S と MyClock.s を同じディレクトリに置いておいてください*3。普通は、これら 4 つのファイルをすべて同じディレクトリに置いておけばよいでしょう。以降、そのように想定して話を進めることにします。

*3:各ファイル中の include 擬似命令で、SXCALL.MAC, WORK.INC のパスを指定する、あるいはアセンブラの/i オプションを使うことによって、どこかのディレクトリからでもインクルードすることは可能ですが、そこまでして別のディレクトリに置く意味はないと思われます。

まずは、各ファイルのアセンブルを行います。

A>AS SKELTON 

A>AS MyClock 

それぞれのファイルが正常にアセンブルできればよし、エラーが発生した場合は、エディタで入力ミスなどをチェックして、エラーの出たファイルを再度アセンブルしてください（ここでエラーの出なかった方には再アセンブルの必要がないこともファイル分割の恩恵です）。

正常にアセンブルできた場合は、

SKELTON.o

MyClock.o

という 2 つのオブジェクトファイルが作成されているはずです。これら 2 つのファイルをリ

ンクして、実行ファイル MyClock.x を作製します。

```
A> LK -O MyClock SKELTON MyClock 
```

正常に終了した場合は MyClock.x が作成されているはずです。この段階でエラーが発生した場合は、SKELTON.S, MyClock.S の xdef, xref 宣言のあたりを調べてみてください。

なお、この作業を make を使って行う場合の makefile は以下のようになります。

```
SKELTON.O:      SKELTON.S WORK.INC SXCALL.MAC
                AS SKELTON.S

MyClock.O:      MyClock.S WORK.INC SXCALL.MAC
                AS MyClock.S

MyClock.X:      SKELTON.O MyClock.O
                LK -OMyClock SKELTON MyClock
```

COLUMN | make について

ファイルを分割してコンパイル/アセンブルを行う場合、厄介なのはファイルの管理であることは本文中で述べました。この問題について、例を挙げて説明しましょう。

A, B, C の3つのファイルで構成されるプログラム ABCがあったとしましょう。A, B, Cそれぞれのファイルをアセンブルした結果、どれも正常にアセンブルを終了しました。ところが、実行してみると、どうも不具合があるようです。どうやら、Bの分担している機能のあたりにバグが潜んでいるらしいと見当をつけ、エディタでチェックしてみると、案の定、些細なミスを発見できました。修正後、内容が更新されたのでBだけをアセンブルし、リンクすると、今度はきちんと動きました。

さて、いまの例は非常にかんたんな例であり、2度目のアセンブルのとき、どのファイルをアセンブルしたらよいかは誰でもわかります。しかし、ファイルが5個も6個もあり、また、一度にいくつものファイルを修正した場合などはどうでしょう？ それが何度も繰り返された場合はなおさらです。どのファイルを再アセンブルしたらよいか？ がすぐにわかりますか？

ディスクにはファイルが作成/更新された日付が記録されています。アセンブルした結果得られるオブジェクトファイルの日付より、ソースファイルの日付のほうが新しければ、そのソースは再アセンブルする必要があると判断できるはずです。人間がこれをいちいちチェックして、手でアセンブルの指示を入力してもよいのですが、それは合理的ではありません。そこで、それを代行してくれるユーティリティ、makeの登場です。

makeが起動されると、とくに指示がないかぎり、makefileというファイルを探します。makefileはテキストファイルであり、makeが行うべき仕事の指示が書かれています。makeはその指示にしたがって、ファイルの日付をチェックし、必要な処理を施します。この「必

要な処理」としてアセンブルを指定しておけば、自動的にファイルの再アセンブルを行ってくれるわけです。

makefile のもっとも基本的な書き方は、以下のとおりです。

```
<ターゲットファイル名> : <比較ファイル名 1> <比較ファイル名 2>...
<コマンド>
```

<ターゲットファイル名>は、一般に<コマンド>によって作成されるファイルの名前です。<比較ファイル名>たちを、<コマンド>によって加工することにより、<ターゲットファイル名>が作成されると考えればよいと思います。そして、<ターゲットファイル名>の日付よりも新しいものが、<比較ファイル名>群の中に1つでもあった場合、<コマンド>が実行されます。逆に、<ターゲットファイル名>の日付が、<比較ファイル名>群の中のどれよりも新しかった場合は何もしません。

先ほどの例でしたら、A.s というファイルから A.o, B.s というファイルから B.o, C.s というファイルから C.o が作成されるのですから、

```
A.o:      A.s
          AS A
                                     ←ここはかならず1行空けます
B.o:      B.s
          AS B
                                     ←ここはかならず1行空けます
C.o:      C.s
          AS C
```

と書いておけば、正しく再アセンブルが行われることになります。

さらに、実行ファイルの生成までも make に行わせることができます。A.o, B.o, C.o をリンクして ABC.X をつくればよいのですから、

```
ABC.X:    A.o B.o C.o
          LK -OABC A B C
```

とすればよいのです。

なお、最終的に得たいファイルを最初にかくのが普通ですから、ABC.X に関する記述は、ほかのファイルに関する記述の前に置きます。

このように makefile を書いておくことによって、一連のアセンブル/リンク作業は

```
A>make 
```

と入力するだけですべて自動で行われてしまいます。

このように便利な make は、C コンパイラ Ver.2.0 に付属してきます。さらに高機能な make としては、おなじみ GNU プロジェクトの GNUmake が移植されています。各ネットのフリーソフトウェアのコーナーで入手できると思いますので、ぜひ使わせてもらいましょう。

1⁴ 実行とデバッグ

サンプルプログラム MyCLOCK は完成しました。しかし、プログラムにバグはつきものです。SX-WINDOW でアプリケーションを起動させるのはかんたんですが、SX-SHELL の上のタスクとして動作するため、デバッグが非常に難しい問題となります。ここでは、前著『SX-WINDOW〜』でも多少触れた、SX-WINDOW アプリケーションのデバッグを解説していきましょう。

SX-WINDOW でファイルを起動するのはかんたんです。ウィンドウを開いて、アイコンをダブルクリックすればよいのですから。この場合、私たちの時計はシェル上で動作することになり、ほかのタスクと同時に時を刻みます (図 1)。

■図 1 シェル上で動作させた場合



もう 1 つの起動方法として、コマンドライン上から直接実行ファイルを起動することが考えられます。私たちの時計で用いたスケルトンは、外部カーネル SXKERNEL.X を呼び出しているため、SXKERNEL.X の環境下で動作します。SXKERNEL.X 上では、私たちの時計だけが動作することになります*1 (図 2)。

*1:正確には、外部カーネルと私たちの時計、計 2 つのタスクが動作しています。

■図 2 外部カーネル上で動作させた場合



以上、2つの方法のどちらでもけっこうですから、時計を立ち上げてみて正常に動作しているかどうかを確認してください。

私たちの時計の場合、正常に動作しているかどうかを確認することは比較的容易です。チェックすべきおまな事項を箇条書きにしてみましょう。

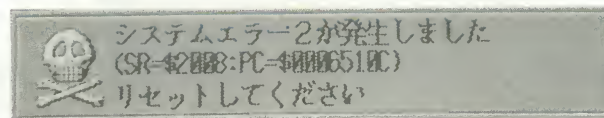
- 1) 起動し、エラーが発生することなく動作するかどうか
- 2) 正しい時刻が、正しく表示されているかどうか
- 3) 1秒おきに時刻の書き換えが行われているかどうか
- 4) コントロール類が正しく動作するかどうか

ほかにも細々としたチェックポイントはいくらでもあります¹、きりがないので、この程度にとどめておきます。

まず最初の問題は、1)の「起動し、動作するかどうか」です。プログラム内で²アドレスエラー、バスエラー等の致命的なエラー（＝システムエラー）が発生した場合、図3のようなダイアログが表示されることがあります。致命的なだけに、この障害がもっとも原因を特定しやすく、デバッグも比較的容易であるともいえます。

*2:このプログラムがSX-SYSTEM、シェル、ほかのタスクに悪影響を及ぼした場合、致命的なエラーがほかのタスク内で発生する場合があります。

■図3 システムエラー



デバッグを行うには、そのためのツールが用意できると好都合です。必要な順に並べてみます。

(1) デバッグ

『Oh!X』誌'90年1月号の付録ディスクに収められていたDB.Xバージョン2.10が用意できれば理想的ですが、それ以前のバージョンでもけっこうです。バージョン2.10以前のDB.Xの場合は、パッチをあてて使用します。このパッチによって、デイスアSEMBル表示時にSXコール名の表示こそできないものの、SXコールのトレースやステップ実行は、ほぼ2.10と同様に行えるようになります。104ページに、前著『SX-WINDOW〜』のコラムに示したパッチ情報を再掲しますので参照してください。

Cコンパイラバージョン2に付属するSCD.Xは、画面の使用状況の関係でSXアプリケーションのデバッグには不向きです。

(2) デバッグ用カーネル SXWDB.X

デバッグを使ってターゲットのプログラムのデバッグを行う場合、通常はターゲットのプログラムだけで、ほかのタスクが動作していない環境下で行うことになります。

外部カーネル SXKERNEL.X のかわりにデバッグ用カーネル SXWDB.X を呼び出すことにより、シェル上で動作しているのと同様な環境で、ターゲットのプログラムを動作させることができます。

SXWDB.X は本書付録ディスク、または『Oh!X』誌 '90 年 1 月号の付録ディスクに収められています*3。

*3:じつをいうと、SXWDB.X は SXWIN.X とまったく同じものです。同じプログラムではありますが、SXWDB.X という名前で実行されることによって、動作が変化します。したがって、SXWDB.X をお持ちでない方は SXWIN.X を複製して、名前を SXWDB.X と変更することによって、手に入れることができます。

(3) コンピュータをもう 1 台

SX アプリケーションを動作させる X68000 と RS-232C ポートどうしをリバーシケールで接続できるコンピュータ（ターミナルと呼ぶことにします）を 1 台用意できると、デバッグはさらに容易になります。コンピュータでなくても、通信機能を備えたワープロでも、もちろんけっこうです。

デバッグは標準出力に表示を行い、標準入力からコマンド等を入力します。SX-WINDOW 上では、標準出力を利用すると、SX-WINDOW の画面が乱れてしまいます。また、普通の状態では、標準入力としてキーボードを利用することはできません。DB.X を /r オプション付きで起動することにより、標準出力、標準入力は、それぞれ AUX 出力、AUX 出力へとリダイレクトされます*4。これによって、画面を乱すこともなく、ターミナル側のコンソールを利用してデバッグを操作することが可能になります。

*4:標準の状態では、AUX 入出力としては RS-232C の入出力が割り当てられています。

こうしたツール類がなくてもデバッグが可能であることはいうまでもありませんが、最低限デバッグは用意したいところです。

各ツールを利用したデバッグの方法を説明する前に、基本となるデバッグの方針を定めておくことにしましょう。SX アプリケーションにかぎった話ではなく、ほとんどすべての環境下でのプログラミングに共通した話ですから、ベテランの方には釈迦に説法かもしれません。

デバッグの第 1 歩は、自分の、あるいは他人のつくったプログラムの処理の内容を把握して、まず大元のアルゴリズムにおかしなところがないかどうかを判断することから始まります。この段階で不具合が発見できた場合は、プログラムを根本から書き直すはめになります。きちんと設計することを怠った報い、といえなくもありません。

アルゴリズムに問題がなさそうだと判断できたところで、おもむろにソースリストを呼び出し、チェックを始めます*5。ここでありがちなのが、タイプミス等の初歩的な間違いです。目

で見て修正できるところは修正して、再アセンブル/コンパイルした結果、まだ不具合がある場合、ここてようやくツール類の登場となります。

*5:この段階でプリンタにソースを出力してしまう人もいますが、資源保護の観点からは、おすすめできることはありません。えてして、この種のプリントアウトは2時間後にはくずかご行きの運命にあります。

1 ツールを利用しないデバッグ

デバッガなどのツールをいっさい用いることなくデバッグを行うことは、もちろん可能です。ことに SX-WINDOW の場合は、デバッグ環境が完備されているとはいいがたい現状なので、ツールに頼らずにデバッグせざるをえない状況に陥ることもないとはいえません*6。

*6:エクセプションマネージャを利用したプログラムなどは、デバッガによるデバッグが難しいケースの1つです。

デバッガを利用するおもな目的は、次の2点に絞られると思われます。

- ・正しい順序で処理が行われているかどうか確かめる
- ・正しい結果、あるいは途中経過が得られているかどうか確かめる

デバッガを利用せずにこれらの目的を達する方法としては、プログラムの状態をなんらかのかたちでユーザに示すためのコードを埋めこむしかありません。よく C や BASIC のプログラムで「怪しい」変数の内容を表示させたり、どこまで処理が進んだかを示す文字列を表示させたりしますが、SX アプリケーションでも同様な手を使おうというわけです。

SX アプリケーションの場合、問題になるのは、どこに、どのように表示するか、です。とりあえず、2つのケースが考えられます。

● Human の標準出力を利用する

前に少し触れましたが、Human の標準出力へ表示を行った場合、SX-WINDOW の画面が乱れてしまいます。ほとんどの場合、動作にはとくに影響がないので、あくまでも美意識や見やすさの問題ですが、この方法の利点は、とにかくかんたんであるということです。

プログラムの適当な場所に、適当な文字列を与えて、DOS コールの `_PRINT` を呼ぶだけでよいのですから、ほとんど何も考えずにすむため、タイピングの手間だけであるといってもいいでしょう。

次のようなケースで、その使い方と効果を示すことにします。

実行例 1

時報設定のチェックボックスをクリックすると、システムエラーが発生してしまう。ほかのコントロールの場合は問題がない

この場合、もっとも怪しいのはレフトダウンイベントの処理ルーチン内であることはすぐにわかります。チェックボックスに関係する部分に注目します。

まず、\$A3A3 `__SXCallCtrlM` が正常に処理を終了しているかどうかをチェックします。コントロールをオープンする際に、異常な引数を指定した場合など、`__SXCallCtrlM` 内でエラーが発生することがあります。そこで、MyClock.S の 96 行目から、次のようなコードを挿入します。

```
pea      DBmsg1(pc)
dc.w     $ff09                      *_print
addq.l   # 4,sp
```

DBmsg1 は固定データ領域に置きます。

DBmsg1:

```
dc.b     ' __SXCallCtrlM finished.',0
```

文字列の末尾に改行 (CRLF) を付加しなかったのは、スクロールが発生させないようにするためです。文字列を表示しただけなら、その文字列の範囲の画面が乱れるだけですみますが、スクロールが発生すると画面全体が乱れてしまいます。

このコードによって、`__SXCallCtrlM` が正常に終了している場合は、画面のどこか*7 に `__SXCallCtrlM finished` の文字列が表示されることになります。すなわち、これは、ここまでプログラムがまがりなりにも実行された、ということを意味しています。

*7: MyClock.x, あるいは SX シェルを起動した際のプロンプトの位置によって、どこに表示されるかが決まります。

次にチェックボックスに関係しているコードは、チェックボックスレコードへのハンドルを収めている `chkBoxHdl` (a5) を参照している 97 行目の `cmp` 命令ですが、「チェックボックスでの」エラーが発生している状況から考えて、ここでの判定は正常に行われていることが考えられます。したがって、エラーが発生しているとする、105 行目からの `MSLD_ChkBox` 以降ということになります。ここでは、システムエラーが発生していることから、109 行目で異常なアドレスにアクセスしていることを疑ってみることにします。`jihou` (a5) の意味するアドレスを 16 進の文字列で標示してみることにして、109 行目から次のようなコードを書き加えることにします。


```

move.l    d0,-(sp)                * D0 の内容を保存

lea       jihou(a5),a0
move.l    a0, d0
lea       strBuf(pc),a0
dc.w      $fe13                  * __HTOS : D0 の内容を 16 進文字
                                   列に

pea       strBuf(pc)
dc.w      $ff09                  * _print
addq.l    #4,sp

move.l    (sp)+,d0               * D0 の内容を復帰

```

strBuf は静的なワークの領域に置きます*8。

*8:ワークエリア等に置いておかまいませんが、一時的な処置でもあることから、ここではもっとも簡便な方法をとることにしました。

strBuf:

```

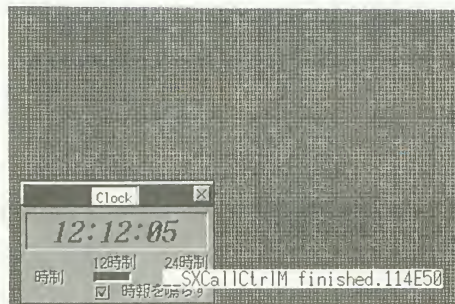
ds.b      10                    * 10 バイトあれば十分？

```

以上のようにコードを追加して、再アセンブル/リンクし、シェル上、または直接コマンドラインから実行します。

正常な状態であれば、チェックボックスをクリックした場合、まず __SXCall CtrlM finished. の文字列が画面を乱しつつ、どこかに表示され、次いで jihou (a5) のアドレスを意味する 16 進文字列が表示されるはず（図 4）。

■図 4 標準出力への文字列表示の結果例



このどちらか、あるいは両方が表示されない場合は、それぞれの箇所を実行する前にシステムエラーが発生しているということを意味していますから、ソースの該当する部分を重点的にチェックします。また、表示される jihou (a5) のアドレスが奇数アドレスだったり、メモ

りの存在しないアドレスだったりするかどうかを調べることで、バグの原因を突き止める手がかりとなります。

両方が表示され、さらにアドレスも正しいのに、それでもシステムエラーが発生する場合は、別の場所に原因があることが考えられます。この場合は、ほかに怪しい場所を求めて、そこで同様に文字列の表示を行って、バグの存在する箇所を特定していくことになります。

● SX-SYSTEM を利用する

同様の方法を、グラフィックマネージャ等を利用して行うこともできます。この場合、画面こそ乱れませんが、標示を行うグラフポートを指定するなどの手間が必要となり、若干面倒ではあります。デバッグ用の一時的なコードに手間をかけたくないのが人情ですから、このようにして文字列等を表示するのはあまりおもしろくありません。

実用的（≡かんたん）なところでは、ダイアログマネージャの\$A2D7 DMBeep 等を利用して、「処理をここまで実行することができた」ことを示す合図としてビーブ音を鳴らすということが考えられます。

2

DB.X バージョン 1.10 を利用するデバッグ

デバッグ用のコードを直接埋め込む方法には若干の問題点があることはすぐに気がつきます。たとえば、怪しい場所がある程度特定できているならともかく、そうでない場合はチェックする場所を変更するたびにソースを書き換えて再アセンブル/コンパイルしなければなりません。埋め込んだコードによって知ることができる以上の情報を得たい場合も同様に、いちいちソースに必要なコードを付け加えなければなりません。要するに、デバッグ作業の自由度が低いことが問題といえます。

Human では、デバッガ（DB.X、SCD.X 等）を使ってターゲットとなるプログラムを監視下に置くことにより、任意のアドレスで実行を停止させたり、レジスタやメモリの内容を参照/変更したりといったデバッグ作業が行えるのはご存知のとおりです。ターゲットが SX アプリケーションの場合も、いくつかの制限がつくことを除けば、ほぼ同様にデバッグ作業を行うことができます。本書執筆時点では、SX-WINDOW 環境で動作するデバッガは存在しないので、制限はあるものの、Human のデバッガを工夫して利用することによって、SX アプリケーションのデバッグを行うことになります。

本書ではデバッガとして DB.X を利用することにしますが、そのほかのデバッガ（GDB、X*⁹等）も利用可能です。ただし、SCD.X は画面全体を破壊してしまうので利用できません。

*9:GNU デバッガ。GNU プロジェクトによって開発されたデバッガで、NIFTY Serve FSHARP の今野氏によって X68000 に移植されている。

以下に DB.X を使って SX アプリケーションのデバッグを行う際の制限を示します。

- ・マルチタスクを前提としたプログラムはデバッグできない

DB.X は、もともと Human 上で動作するプログラムをデバッグすることを目的としてつくられています。そのため、原則として 1 つのプログラム (タスク) だけが動作している状態でしかデバッグを行うことができません。

たとえば、2 つのタスクが連絡を取りあって動作したり *10、シェル上で動作しなければ意味がなかったりする *11 ような SX アプリケーションのデバッグはできないと考えてください。

*10:ただし、通信相手のタスクを自分で起動するような SX アプリケーションならば、ある程度動作の確認は可能です。

*11:例としては、ファイルのアイコンがドラッグされてくることによって動作するようなプログラムや、タスクマネージャスクラップの内容を参照/操作するようなプログラムなどが考えられます。

こうした点については、デバッグ用のコードを埋め込む方法のほうが有利です。なにしろ、SX アプリケーションそのものであるわけですから、SX シェルという環境下で、そのまま起動し、利用することができるからです。

この制限は、SXWDB.X を利用することによって、ある程度緩和されます。

- ・画面を破壊してしまう

DB.X の表示は標準出力に行われます。すでに述べたように、標準出力に文字を出力すると、SX-WINDOW の画面は乱れてしまいます。とくにスクロールが発生すると、画面全体が乱れてしまうことになります。

また、DB.X のコマンド等は標準入力から受け付けていますが、SX-WINDOW ではキーボードマネージャの関係で標準入力は使用できない状態が普通です。デバッグにコマンドを与える場合は、一度、[CTRL] + [OPT.1] + [F10] を押すことで、OldOn を 1 にしておく必要があります。

この制限は、ターミナルを用意することによって回避することができます。

それでは、こうした制限の下でデバッグを使ってみましょう。

実行例 2

ウィンドウレコードの内容がどのように変化するか調べてみたい。

MyClock.x は、すでに作成してあるものとします。リンクの際、リンクに /x オプションを指定せず、グローバルシンボルを残しておくと、プログラム内部を探索する助けになります。まずは、デバッグを起動します。

```
A>DB MyClock.x
```

デバッガの起動と、MyClock.x の読み込みが正常に行われた場合は、次のように表示され、デバッガのコマンド待ちとなります。

```
X68k Debugger v2.10 Copyright 1987, 88, 89 SHARP/Hudson
loading MyClock.x
PC=000FEF00 USP=000DD624 SSP=000067F2 SR=0000 X:O N:O Z:O V:O C:O
D 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A 000FEDE0 000FF590 000DDFB4 0005CF00 000FEF00 00000000 00000000 000DD624
lea $0040(A1), A1 ;000FF5D0(14)
-
```

(各レジスタの内容は、起動する際的环境によって異なります)

ウィンドウを開いた直後にブレークポイントを仕掛け、そこでウィンドウレコードへのポインタを調べることにします。ウィンドウを開いているのは初期化ルーチン `_INIT` の中ですから、シンボル `_INIT` からをディスアセンブル表示して、目的の場所を探します。

```
-l _INIT
_INIT:
000FF14C move.l $0008(A5), D0
000FF150 move.w $0010(A5), D1
000FF154 btst.l #$0000, D1
000FF158 beq.s $000FF174
000FF15A move.l $000C(A5), D1
000FF15E beq.s $000FF17A
000FF160 tst.w D1
000FF162 cmp.w D0, D1
-|
: (何度か繰り返して)
-|
000FF1D0 move.w $FFFF, -(A7)
000FF1D4 move.l $FFFFFFFF, -(A7)
000FF1DA move.w $0200, -(A7)
000FF1DE move.w $FFFF, -(A7)
000FF1E2 pea $01DA(PC)
000FF1E6 pea $0008(A5)
000FF1EA pea $002A(A5)
000FF1EE _WMOPEN ←Ver2.10以前では"dc.w $A1F9"と表示されます
-|
000FF1F0 lea $001A(A7), A7
000FF1F4 tst.l D0
000FF1F6 bmi.w $000FF294
000FF1FA st $009C(A5)
000FF1FE clr.l $00AC(A5)
000FF202 clr.l -(A7)
000FF204 move.w #$0020, -(A7)
000FF208 move.w #$0001, -(A7)
-
```

ウィンドウレコードアドレスは、`$A1F9 WMOpen` の返り値として `A0` に収められています。そこで、`WMOpen` から戻ってきた直後にブレークポイントを仕掛けることにします。ただし、`SX` コールの直後の命令はトレースできない場合があるので、さらに次の命令、例に示した場合では、`FF1F4` にブレークポイントを設定します。

-b FF1F4

この後、G コマンドで MyClock.x の実行が開始されます。デバッガから立ち上げた場合、コマンドラインから立ち上げたのと同じことになりますから、処理が始まるのはラベル DiXstart からです。外部カーネルが起動され、最小限の初期化が行われた後、MyClock 本来の処理が始まります。

普通ならば、このまま時を刻むところですが、先ほど設定したブレークポイントのためにウィンドウを開いた直後に停止します。ブレークポイントに達して処理が中断された場合は、その時点でのレジスタの内容等が標準出力に表示されるため、図 5 に示したように画面が乱れてしまいます。

■図 5 デバッグ中の画面の例 (1)



画面の乱れは実行には直接影響しないので、あまり気にせず作業を続けることにします。

ここで表示されているレジスタの内容のうち、AO レジスタがウィンドウレコードの先頭アドレスを意味しているので、メモ等に記録しておきます。

中断している処理を継続させるために、G コマンドを入力したいところですが、このままではキー入力できません。[CTRL] + [OPT.1] + [F10] を押して、標準入力を有効にしてから、G コマンドを入力してください。

この後、ウィンドウレコードの内容を確認したい場合は、インタラプトスイッチで処理を中断させて、D コマンド等を使います。D コマンドの表示などでスクロールアップが発生すると、テキスト VRAM の #0, 1 ページだけがスクロールアップしてしまうため、ウィンドウ等もスクロールアップしてしまうように見えますが、実際の位置は変化していません。ウィンドウが見えなくなってしまうと、ウィンドウのドラッグリージョンのあたりをマウスでクリックすることで輪郭が確認できますから、一度画面の外にドラッグする等して、アップデートを発生させればふたたび表示されます。

処理を終える場合は、かならずウィンドウのクローズボタンを押して終了処理を行わせてから、デバッガの Q コマンドでコマンドラインに戻ってください。中断した状態からいきなり Q コマンドで終了すると、SX-SYSTEM の終了処理が行われていないため、いろいろと不都合が発生します。

3

SXWDB.X を利用するデバッグ

SXWDB.X は SXKERNEL.X と同様、外部カーネルの一種で、デバッグに直接的に関係するものではありません。が、DB.X 等と組み合わせることによって、SX アプリケーションをシミュレーションに近い環境で動作させながらデバッグを行うことができるようになります。

SXWDB.X の利用方法は、SXKERNEL.X と同様、つまり、コマンドラインから立ち上げられた場合に、まず SXWDB.X を起動し、SXWDB にターゲットの起動をゆだねる、というものです。このためには、スケルトンを差し替えて、再アセンブル/リンクする必要があります。差し替えるべきデバッグ用スケルトン SKELDB.S をリスト 1 に示します。

■リスト 1 SKELDB.S

```

1  *
2  *
3  *
4  *
5
6      .include      DOSCALL.MAC
7      .include      SXCALL.MAC
8
9      .xref      _INIT,_TINI
10     .xref      IDLE
11     .xref      MSLDOWN,MSLUP,MSRDOWN,MSRUP
12     .xref      KEYDOWN,KEYUP
13     .xref      UPDATE,ACTIVATE
14     .xref      SYSTEM1,SYSTEM2,SYSTEM3,SYSTEM4
15
16     .include      WORK.INC      * ワークエリアの内容
17                                     * を定義するファイル
18
19     .text
20 mdhead:
21     dc.l      'OBJR'
22                                     * [ モジュールヘッダ ]
23                                     * R型モジュール
24     dc.l      0
25                                     * プログラムエリアの
26                                     * サイズ (Xファイルの
27                                     * 場合意味がない)
28     dc.l      _main-mdhead
29                                     * スタートアドレスオフセット
30     dc.l      WORKSIZE+STKSIZE
31                                     * ワークエリアのサイズ
32     dc.l      0,0,0,0
33                                     * システム予約
34
35 DiXstart:
36                                     * コマンドラインから
37                                     * 起動した場合
38                                     * ここからスタートする
39
40     lea        64(a1),a1
41     move.l     a1,sp
42     lea        16(a0),a0
43     sub.l      a0,a1
44     move.l     a1,-(sp)
45     pea        (a0)
46     DOS        _SETBLOCK      * 専有メモリを縮小する
47
48     clr.l      -(sp)
49     pea        comm(pc)
50     pea        shname(pc)
51     move.w     #2,-(sp)
52     DOS        _EXEC
53                                     * デバッグ用カーネルの
54                                     * バスをサーチする

```

```

41      clr.l    -(sp)
42      pea     comm(pc)
43      pea     shname(pc)
44      clr.w    -(sp)
45      DOS     _EXEC                                # デバッグ用カーネルを
                                                    # 立ち上げる

46
47      tst.l    d0                                # 正常に終了した場合
48      bpl     p_execil                           # そのまま終了
49
50      pea     mes_execerr(pc)                    # エラーメッセージを
51      DOS     _PRINT                             # 表示する
52 p_execil:
53      DOS     _EXIT                                # 終了
54
55      .data
56 mes_execerr:
57      dc.b     'カーネルの起動に失敗しました!!!', 13, 10, 0
58      .even
59 shname:
60      dc.b     'sxwdb.x -D -K -L0', 0           # デバッグ用カーネルの名前
61      ds.b     70
62      .even
63
64      .bss
65 comm:
66      ds.b     258
67
68
69      .text
70 _main:
71
72      movea.l  a1, a5
73      move.l   a2, cmdLine(a5)
74      move.l   a3, envPtr(a5)
75
76      clr.w    -(sp)
77      clr.l    -(sp)
78      pea.l    winRect(a5)
79      pea.l    (a2)
80      SXCALL   $A3EA                                # _TSTakeParam
81      lea.l    14(sp), sp                          # コマンドラインを解析し、'-W
82      move.w    d0, paramFlg(a5)                   # オプションを得る
83
84      bsr      _INIT                                # アプリケーションの初期化
85
86      bmi     _exit                                # 初期化時に
87
88
89      move.w    #ffff, eventMask(a5)               # エラーがあれば終了
90
91      loop:
92      pea     eventRec(a5)
93      move.w    eventMask(a5), -(sp)               # メインループ
94      SXCALL   $A357                                # _TSEventAvail
95      addq.l    #6, sp                              # イベントを得る
96      lea     eventTable(pc), a1
97      move.w    eventRec_what(a5), d0
98      and.w    #15, d0
99      add.w    d0, d0
100     move.w    (a1, d0.w), d0                      # イベントコードによって
101     jsr      (a1, d0.w)                            # 分岐する
102     tst.l    d0
103     bmi     _exit
104     bra     loop

```


```

101
102 eventTable:
103         dc.w    IDLE-eventTable      # 分岐先のテーブル
104         dc.w    MSLDOWN-eventTable   # 0   アイドルイベント
105         dc.w    MSLUP-eventTable      # 1   レフトダウンイベント
106         dc.w    MSRDOWN-eventTable    # 2   レフトアップイベント
107         dc.w    MSRUP-eventTable      # 3   ライトダウンイベント
108         dc.w    KEYDOWN-eventTable    # 4   ライトアップイベント
109         dc.w    KEYUP-eventTable       # 5   キーダウンイベント
110         dc.w    UPDATE-eventTable     # 6   キーアップイベント
111         dc.w    DAMMY-eventTable      # 7   アップデートイベント
112         dc.w    ACTIVATE-eventTable    # 8   -
113         dc.w    DAMMY-eventTable      # 9   アクティブイベント
114         dc.w    DAMMY-eventTable      # 10  -
115         dc.w    SYSTEM1-eventTable     # 11  -
116         dc.w    SYSTEM2-eventTable    # 12  システムイベント 1
117         dc.w    SYSTEM3-eventTable    # 13  システムイベント 2
118         dc.w    SYSTEM4-eventTable    # 14  システムイベント 3
119
120 DAMMY:
121         rts
122
123 _exit:
124         bsr      _TINI                  # [ 終了する ]
                                           # アプリケーションの
                                           # 終了処理
125
126         move.w   d0, -(sp)
127         SXCALL   $A352                  # __TSExit
128
129         .end     DiXstart
130

```

私たちの時計の場合ならば、再アセンブル/リンクは次のように行うことになります。

A> AS SKELDB 

A> AS MyClock  (すでにアセンブルしてある場合は不要)

A> LK -O MyClock SKELDB MyClock 

make を使う場合は、次のような makefile を書きます。

```

MyClock.X:      SKELDB.O MyClock.O
               LK -OMyClock SKELDB MyClock

SKELDB.O:       SKELDB.S WORK.INC SXCALL.MAC
               AS SKELDB.S

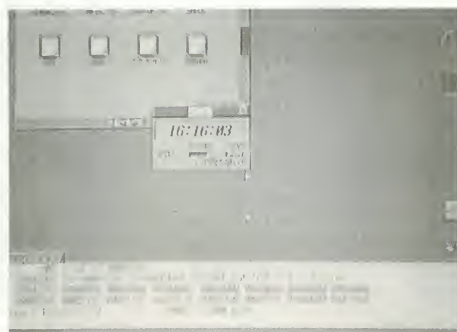
MyClock.O:      MyClock.S WORK.INC SXCALL.MAC
               AS MyClock.S

```

このように作成した実行ファイルを、デバッガによってチェックする手順は、先ほどと変わりません。が、起動したときの画面 (図 6) を見ると、先ほどとは異なり、シェルとほとんど同じであることがわかります。ディレクトリのウィンドウを開いたり、ファイルを実行できたり、そして、それがマルチタスクで動作したりするのもシェル上と同じです。ただし、ブレー

クポイントに到達したり、インタラプトスイッチを押したりして処理を中断させた場合は、すべてのタスクは停止し、デバグがだけが動作する状態になります。

■図6 デバグ中の画面の例(2)



デバグの表示によって画面が壊れたり、終了時に注意が必要だったりする点は、SXWDB.Xを使わない場合と同様です。キー入力に関する注意は、SXWDB.Xに-Kオプションをつけて標準入力を有効にしているため、必要なくなっています。

4 ターミナルを利用するデバグ

これまでの方法で問題だったのは、とにかく画面が乱れてしまうことでした。この問題は、ターミナルとなるパソコン/ワープロが用意できれば解決できます。

ターゲットのプログラムを実行する X68000 と、ターミナルを接続する手順は以下のとおりです。

- 1) X68000 とターミナルの RS-232C ポートをクロスケーブルで接続する
- 2) X68000 側で SPEED.X を実行して、ボーレート等を設定する
- 3) ターミナル側で通信ソフトを立ち上げ、ボーレート等を X68000 側と合わせる

以上で準備は完了です。

● デバグを利用しない場合

デバグなしでデバグを行う場合は、標準出力に文字列を表示する等のデバグ用のコードを埋め込んで再アセンブル/コンパイルした後、

A> MyClock>AUX 

のように、標準出力を AUX にリダイレクトして起動します。これによって、画面を乱しつつ表示されるはずだった文字列は、ターミナル側に表示されるようになります。

● デバッガを利用する場合

デバッガをリモートデバッグモードで起動することにより、デバッガの表示やコマンドの入力等をターミナル側で行えるようになります。リモートデバッグモードでデバッガを起動するには、/r オプションを指定します。

```
A>DB /r MyClock.x 
```

SXWDB.X を併用することによって、非常に快適なデバッグが可能となります (図 7)。

■図 7 リモートデバッグ中



SX アプリケーションにかぎらず、デバッガの表示やキー入力に問題があるようなプログラム (ゲーム等) では、こうしたリモートデバッグはごく一般的に使われている方法です。サブマシンをお持ちの方はぜひお試しください。

以上に示したような方法を、場合に応じて組み合わせて使い、バグの原因を 1 つ 1 つ潰していきます。デバッグ作業の「地味さ」は、Human のプログラムであろうと、SX アプリケーションであろうと変わるわけではありません。マルチタスクで動作するデバッガがないというのも痛いところですが、ほかのタスクのことを考えずにすむので、かえって好都合かもしれません。

また、デバッガを使って SX アプリケーションを追っていくことで、SX-SYSTEM への理解をさらに深めることができるはずです。ひととおりバグの退治ができた後は、SX-WINDOW 内部を探索してみてもいいかもしれません。

COLUMN DB.X へのパッチ当て

DB.X にはいくつかバージョンがありますが、このうち、もっとも SX-WINDOW 用のアプリケーション開発に向いているのはバージョン 2.10 です。『Oh! X』誌に付属してきた 2.10 では、SX コールが実行可能で、ディスアセンブル時に SX コール名まで表示してくれます*。

C コンパイラ 2.0 に付属する DB.X バージョン 2.0 は、SX コール名の表示こそできませんが、SX コールの実行は可能です。

問題はそれ以前のバージョン、1.00 と 1.01 で、これらは SX コールを実行しようとするエラーが発生してしまいます。これらには、以下のようにパッチをあてて使用してください。

バージョン番号	サイズ	日付	時刻	オフセット	旧データ	新データ
1.00	25522	87-11-03	12:00:00	\$4DFE	7008	7007
1.01	25518	88-05-15	12:00:00	\$4DFA	7008	7007

*：バージョン 2.10 では SX1.02 までの SX コールのみサポートされています。

拡張されたマネージャ

SX-WINDOW バージョン 1.10 を使ってみて最初に感じた「速さ」は、従来から用意されていたマネージャのコードを見直し、全体的にスピードアップを図った結果です。しかし、高速化ばかりでなく、機能の面でも大幅に拡張されています。この章では、とくに大きな拡張が行われたグラフィックマネージャ、テキストマネージャ、タスクマネージャを中心に、それぞれの新しい機能/概念等を解説します。

2¹ グラフィックマネージャ

SX1.10とSX1.02を比較した場合、すぐに気がつくのが全体的な描画スピードの向上です。これは、ウィンドウをはじめ、画面に表示されるあらゆるものを描画しているグラフィックマネージャが改善されたのが大きな理由です。こういったスピード的な改善もさることながら、機能的にもさらに充実してきました。

1 スクリーンタイプの追加

従来、スクリーンタイプとしてはテキストタイプとグラフィックタイプの2種類が用意されていましたが、GR2タイプ、GR3タイプの2つのスクリーンタイプが追加されました。

GR2、GR3は垂直型（グラフィック画面同様に、ドットの色を一度に指定できるタイプ）のビットマップです。これらに対応するハードウェア的なディスプレイ装置はX68000には備えられていないので、表示を行う目的ではなく、画像データをメモリ中に記憶しておく目的で利用されます。

GR2タイプでは、1ドットを4ビット、1バイトで2ドットを表現します。したがって、16色まで表現可能です。

GR3タイプでは、1ドットを8ビット、1バイトで1ドット、256色までを表現します。

これら2つが追加されたことにより、ビットマップの種類を示すコードは次の表のように拡張されました。

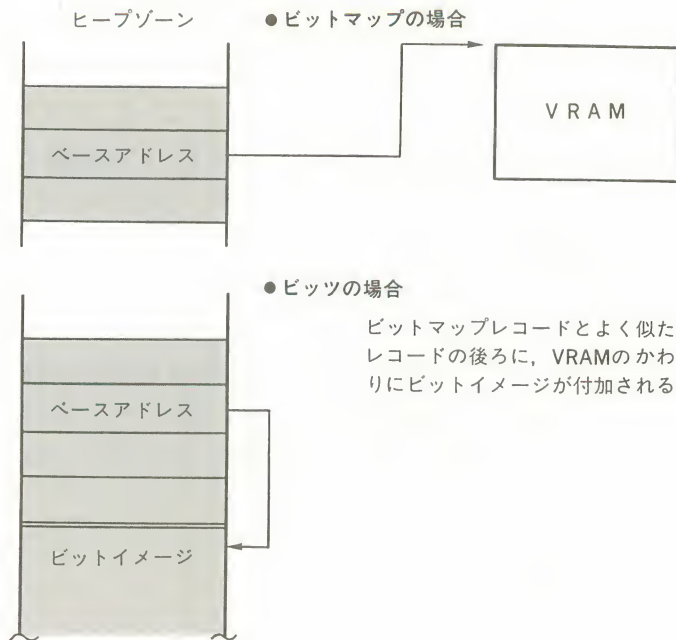
コード	スクリーンタイプ
0	テキストタイプ
1	グラフィックタイプ
2	GR2タイプ
3	GR3タイプ

ただし、\$A12D GMOpenGraph などではGR2、GR3タイプは指定できません。GR2、GR3タイプを指定できるのは、次に解説する「ビット」の作成の場合のみと考えてください。

2 ビットマップのバリエーションの追加

従来のビットマップに加えて、メモリ中に画像情報を保存しておくためのビットマップのバリエーションとして、「ビット (bits)」と呼ばれるデータ構造が追加されています。ビットはビットマップレコードと再配置可能ブロックを結合したようなもので、再配置可能ブロックの中にビットマップレコードと画像情報が格納されます (図 1)。

■図1 ビットの概念



ビットの形式は次のとおりです。

オフセット	欄 名	内 容	
+\$00.w	bmKind	ビットマップの種類	(1)
+\$02	bmRect	ビットマップレクタングル	
+\$0a.l	bmBase	ベースアドレス	(2)
+\$0e.w	bmLine	1ラインのバイト数*1	
+\$10.l	bmPage	1ページのバイト数	} テキストタイプの場合
+\$14.w	bmAPage	アクセスページ	
+\$10.w	bmBRatio	ブレンドウェイトレシオ	} グラフィック, GR2, GR3 タイプの場合
+\$12.l	- - -	未使用	
+\$16.l		イメージサイズ	(3)
+\$1a.w		ロックサイズ	(4)
+\$1c		システム予約 (16 バイト)	
+\$2c		ビットイメージ	(5)
⋮		⋮	

*1:「SX-WINDOW～」ではグラフィックタイプの場合、1ラインのバイト数がロングワードで示されるような記述がありましたが、それは誤りでした。

(1) ビットマップの種類

ビットマップレコードと同じように、ビットマップの種類が格納されています。ビットマップではテキストタイプ、グラフィックタイプのみが指定可能ですが、ビットでは GR2, GR3 タイプも指定することができます。

(2) ベースアドレス

ビットマップの場合、テキスト VRAM, あるいはグラフィック VRAM のアドレスが格納されますが、ビットでは (5) のビットイメージのアドレス (つまり、ビットのオフセット + \$2c), もしくは 0 が格納されています。

ビットは再配置可能ブロック内にあるため、通常はビットイメージのあるアドレスを確定することができません。そのため、確定できない場合には 0 が収められ、この間、ビットに対する描画などは不可能です。ビットを利用する場合は、\$A1CC GMLockBits を利用してビット全体をロックし、再配置による移動を禁止します。このとき、ビットイメージのアドレスも確定でき、ベースアドレスに格納されます。この状態で、はじめてビットに対して描画などを行うことができます。

(3) イメージサイズ

ビットイメージのバイト数が収められています。

(4) ロックレベル

1 以上が収められている場合は、ビットはロックされた状態であることを、0 の場合はロックされていないことを意味しています。\$A1CC GMLockBits, \$A1CD GMUnlockBits 等で操作されます。

(5) ビットイメージ

ビットの画像データが収められています。

ビットの利用は、次のような手順で行います。

① ビットの作成

\$A1CA GMNewBits によって、必要なサイズのビットをヒープゾーン中に作成します。このとき、ビットへのハンドルが返されます。

この時点では、ビットはロックされておらず、また、ビットイメージの内容も初期化されていません。

② ビットのロック

ビットに描画を行う場合は、まずロックを行わなければなりません。ビットへのハンドルを指定して \$A1CC GMLockBits を呼ぶことにより、ロックが行われます。

③ 内容の描画

ビットへの描画は、ビットマップへの描画とまったく同様に行うことができます。ビットマッププレコードを参照して、直接ビットイメージを操作することもできますが、このビットのためのグラフポートを作成して、グラフィックマネージャの SX コールを利用するほうがよかったです。

④ ビッツのアンロック

描画が一段落したところで、\$A1CD GMUnlockBits を呼び出してビットをアンロックします。ビットも再配置可能ブロックの一種なので、長期にわたってロックを行うことはヒープゾーンの利用効率を低下させる可能性があります。

⑤ ビッツの内容の利用

ビットに描画を行った結果を、なんらかの方法で利用できなければ意味がありません。

ビットの内容を実際に表示しようとする場合、ビットとビットマップ間でスクリーンタイプが異なる場合が考えられますが、このような場合のために \$A1BB GMTransImg のように、異なるスクリーンタイプのビットマップ（ビット）間で、データ変換を行いつつコピーする機能等が用意されています。

⑥ ビッツの廃棄

ビットの利用がすべて終了したら、アプリケーションは責任をもってビットを廃棄しなければなりません。ビットの廃棄を行う場合は \$A1CB GMDisposeBits を呼び出します。

3

図形の追加

グラフィックマネージャの扱う図形の種類に、「円弧 (arc)」、「ポリゴン」の 2 種類が正式に追加されました*2。

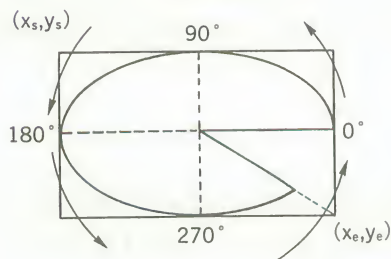
*2: SX1.02 でも、いちおうの描画ルーチンは用意されていた模様です。

① 円弧 (arc)

円弧は、楕円のある一部分を切り取った形です。円弧の指定は、レクタングル、および開始角度、終了角度の 3 つで行われます。グラフィックマネージャは、指定されたレクタングルに内接する楕円の、開始角度から終了角度までの部分を描画します (110 ページ図 2)。

■図2 円弧の指定

(a) レクタングルと角度の意味



このような円弧は、
レクタングル
 $(x_s, y_s) - (x_e, y_e)$
と
開始角度\$0000(0)
終了角度\$13B(315)
で表される

(b) 枠を描画した場合



(c) 内部を塗り潰した場合



② ポリゴン

ポリゴンは複数の頂点をラインで結んだ、いわゆる多角形です。ポリゴンは頂点を列挙することによって表現します。頂点は列挙された順番にラインで結ばれ、最後の頂点と先頭の頂点とが結ばれた、閉じた図形となります (111 ページ図3)。

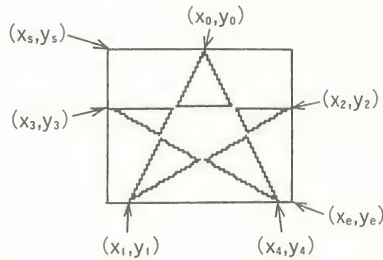
ポリゴンを表するポリゴンレコードの形式は次のとおりです。

オフセット	内 容
+\$00.l	ポリゴンレコードのサイズ
+\$04	バウンドレクタングルを意味するレクタングル
+\$0c.l	最初の頂点を意味するポイント
⋮	⋮

ポリゴンの描画には、枠の描画と内部を塗り潰しての描画がありますが、塗り潰しの場合、「内部」の定義がリージョンのそれに準ずる点に注意してください。

■図3 ポリゴンの指定

(a) 枠を描画した場合



このようなポリゴンは

```
dc.l 4+8+(4*6)
dc.l x_s-y_s,x_e-y_e
dc.l x_0-y_0
dc.l x_1-y_1
dc.l x_2-y_2
dc.l x_3-y_3
dc.l x_4-y_4
dc.l x_0-y_0
```

(b) 内部を塗り潰して描画した場合



4 多様な画面モードへの対応

SX1.02 は、基本的に 768×512 ドットの画面モードで動作することが前提となっており、それ以外の画面モードのことは考えられていませんでした。

SX1.10 では、グラフィックマネージャの初期化時に \$A1C2 GMinInitGrapMode を呼び出して画面モードを指定することにより、ビットマップの初期値等を画面モードに適した値に変更します。

5 フォントの拡張

SX1.02 では、原則として ROM に格納されているフォントしか利用することができず、フォントカインドとしては ROM 中の 12, 16, 24 ドットフォントを意味する 0, 1, 2 の値しか取り得ませんでした。

拡張されたグラフィックマネージャでは、「フォントリンク」というデータ構造を使うことによって、ユーザがフォントを追加できるようになっています。フォント 1 種類ごとにフォントリンクを用意し、それへのポインタを指定して、\$A1E0 GMAAddFont を呼び出すことにより、SX-SYSTEM にフォントが追加されます。追加されたフォントは、フォントリンク中で宣言したフォントカインドの数値で利用することができます。

フォントリンクの形式は次のとおりです。

オフセット	欄名	内 容	
+\$00.l	flNext	次のフォントリンクへのポインタ	(1)
+\$04.w	flKind	フォントカインドの値	(2)
+\$06.l	flName	フォント名 (ASCIIZ) へのポインタ	(3)
+\$0a.l	flProc	フォントプロセスのアドレス	(4)
+\$0e.l	flMem	フォントが利用する DOS メモリブロックへのポインタ	(5)
+\$12.l	flRsv	システム予約	

(1) 次のフォントリンクへのポインタ

システムが使用するので、内容を変更してはいけません。また、あらかじめ内容を設定しておく必要はありません。

(2) フォントカインドの値

ここに収めた値をフォントカインドとして指定することにより、このフォントを利用して文字列の描画等が行われます。3 以上の値を指定してください。また、1 つのフォントカインドには 1 つのフォントのみが対応します。

(3) フォント名へのポインタ

ASCIIZ 型で表現されたフォント名の文字列へのポインタが入ります。

(4) フォントプロセスのアドレス

フォントを描画したり、フォントに関する計算を行ったりするルーチンのアドレスを収めます。フォントプロセスは、DO にコマンドコードが、AO にはパラメータポインタが格納された状態で呼び出されます。フォントプロセスの中では DO 以外のレジスタは破壊してはいけません。

フォントプロセスを設定しないときにはここに 0 を収めますが、その場合はプロセスポインタが登録されている、対応する描画ルーチンが呼び出されます。

フォントプロセスのコマンドを以下に示します。

●command=0: FP_INIT

パラメータ なし

返り値 DO.l =0 正常終了
 ≠0 異常終了

フォントの初期化を行います。

このコマンドは SX-SYSTEM から呼び出されません。必要な場合は、フォントを追加したタスク等が呼び出してください*3。

*3: もとものの仕様では \$A14D GMInitialize で呼び出される予定だったようです。

●command=1:FP_TINI

パラメータ なし

返り値	DO.1	=0	正常終了
		≠0	異常終了

フォントを開放する際の処理を行います。

●command=2:FP_INFO

パラメータ なし

返り値	DO.1	=0	正常終了
		≠0	異常終了

1 文字の全角フォントの幅と高さの最大値を求めます。

\$A198 GMFontInfo から呼ばれます。

●command=3:FP_DRAW (プロセスポインタ+\$00 に相当)

パラメータ	(AO).1	文字列へのポインタ
	4(AO).1	文字列先頭からのオフセット
	8(AO).w	文字列のバイト数

返り値	DO.1	=0	正常終了
		≠0	異常終了

文字列の描画を行います。

●command=4:FP_LENGTH (プロセスポインタ+\$30 に相当)

パラメータ	(AO).1	文字列へのポインタ
	4(AO).1	文字列先頭からのオフセット
	8(AO).w	ドット数

返り値	DO.1	文字列のバイト数
-----	------	----------

指定したドット数内に収まる文字列のバイト数を求めます。

●command=5:FP_WIDTH (プロセスポインタ+\$2c に相当)

パラメータ	(AO).1	文字列へのポインタ
	4(AO).1	文字列先頭からのオフセット
	8(AO).w	文字列のバイト数

返り値	DO.1	上位ワード, イタリックを含めた幅 (ドット数)
		下位ワード, イタリックを含めない幅 (ドット数)

指定した文字列の幅を求めます。

- command=6 : FP_REV
フォントの戻り幅を返します。
現在は使用されていません。
- command=7 : FP_RSV
システム予約です。

(5) フォントが利用する DOS メモリブロックへのポインタ

フォントのデータが収められている DOS のメモリブロックへのポインタを収めておく、フォント開放時に DOS コールの MFREED によって自動的に開放されます。その必要がない場合は 0 を収めておきます。

フォントの拡張にともなって、現在のフォントに関する情報をまとめて取得する SX コール、\$ALC3 GMCurFont が追加されています。これによって得られる情報は、フォントレコードというかたちで返されます。フォントレコードの形式は以下のとおりです。

オフセット	欄 名	内 容
+\$00.w	fmKind	現在のフォントのフォントカインド
+\$02.w	fmFace	現在のフォントのフォントフェイス
+\$04.l	fmSize	現在のフォントのフォントサイズを示すポイント
+\$08.l	fmWork	現在のフォントの内部データ
+\$0c.l	fmRev	戻り幅の最大値 (固定小数点数)
+\$10.l	fmWidth	全角文字幅の最大値 (固定小数点数)
+\$14.l	fmWhalf	半角文字幅の最大値 (固定小数点数)
+\$18.l	fmHeight	文字の高さ (固定小数点数)
+\$1c.w	fmExtra	イタリック時の追加幅
+\$1e.l	fmXzoom	X 方向拡大率 (固定小数点数, 標準なら 0)
+\$22.l	fmYzoom	Y 方向拡大率 (固定小数点数, 標準なら 0)

6 疑似ダイアログ

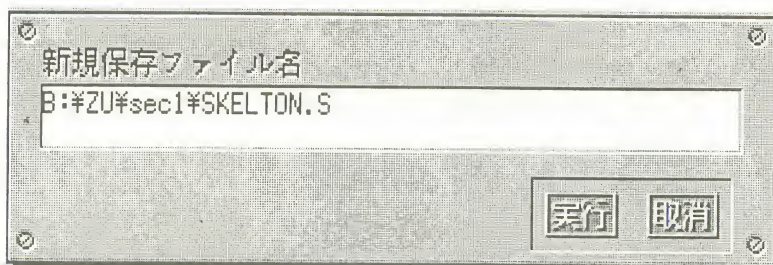
ユーザからの入力を受け付ける場合、ダイアログは非常に重宝する反面、ほかのタスクが停止してしまうという問題がありました。ちょっとした文字列の入力など、ほかのタスクを止めてまでユーザの注意を集中させるまでもない場合のために、疑似ダイアログが用意されました。

疑似ダイアログはエディタ、x の中で、ファイル名の入力などに用いられています (図 4)。

ダイアログという名前がついてはいますが、これはビットマップに描画される図形の一種と考えられます。結局のところ、カレントビットマップに表示される長方形の図形でしかありません。したがって、疑似ダイアログの使い方はダイアログ等とはまったく異なります。

ウィンドウコンテンツの適当な位置に疑似ダイアログを描画したら、その内部にコントロールやテキストエディットを置いて、ユーザからの操作に対応します。このコントロール等は、結局ウィンドウコンテンツ上にあるだけですから、その処理になんら特別な手法は必要とされ

■図 4 疑似ダイアログ



ません。

リターンキーが押されるなり、ボタンが押されるなりして疑似ダイアログ上での操作が終了したら、コントロール等を廃棄してウィンドウの内部を書き直せば、疑似ダイアログに関する処理は完了です。

7 下位ルーチンのユーザへの開放

グラフィックマネージャの描画ルーチンやフォントの描画ルーチンなど、下位の描画ルーチンをアプリケーション作成者がつくりことができるようになったわけですが、それを支援するために、SX-SYSTEM 内の下位ルーチンを利用することができるようになりました。

\$A1E3 GMGetHProcTbl, \$A1E6 GMGetStdProcTbl, \$A1E7 GMGetFontProcTbl, \$A1E8 GMGetRgnProcTbl は、それぞれ水平描画の初期化ルーチンのテーブル、標準描画ルーチンのテーブル、文字描画ルーチンのテーブル、リージョン 1 行演算ルーチンのテーブルを得るための SX コールです。テーブルを参照して、必要な引数を添えて呼び出すことによって、これら下位のサブルーチンを利用することができます。

これらのテーブルは、いずれも SX-SYSTEM 内部にありますから、変更することはできません。

それぞれのテーブルの形式と、そこに登録されている下位ルーチンについて、かんたんな機能と、引数、返り値を示すことにします。

●水平描画の初期化ルーチンのテーブル (\$A1E3 GMGetHProcTbl)

オフセット	欄 名	内 容
+\$00.l	hLineInit	水平ライン描画のための初期化ルーチン
+\$04.l	hCopyInit	水平ラインコピーのための初期化ルーチン
+\$08.l	hPutInit	1 ラインの描画を行うための初期化ルーチン

(1) hLineInit：水平ライン描画のための初期化ルーチン

引数 AO.1 水平ライン描画用ワークへのポインタ

- (AO).1 1 ライン描画サブルーチンのアドレス
 - 4(AO).1 0 ページ描画ルーチンエントリアドレス
 - 8(AO).1 1 ページ描画ルーチンエントリアドレス
 - 12(AO).1 2 ページ描画ルーチンエントリアドレス
 - 16(AO).1 3 ページ描画ルーチンエントリアドレス
 - 20(AO).w パターンの Y オフセット
 - 22(AO).w パターンの 1 行のバイト数
 - 24(AO).w パターンオフセットのマスクデータ
 - 26(AO) 縦横 16 ドットのパターン
- : (スクリーンタイプによって内容・サイズは変化)

A5.1 描画を行うグラフポートレコードへのポインタ

返り値 DO.1 リザルトコード

(2) hCopyInit : 水平ラインコピーのための初期化ルーチン

引数 DO.w コピーモード

AO.1 水平ラインコピー用ワークへのポインタ

- (AO).1 1 ラインコピーサブルーチンのアドレス
- 4(AO).1 0 ページコピールーチンエントリアドレス
- 8(AO).1 1 ページコピールーチンエントリアドレス
- 12(AO).1 2 ページコピールーチンエントリアドレス
- 16(AO).1 3 ページコピールーチンエントリアドレス
- 20(AO).w 0 ページ反転データ
- 22(AO).w 1 ページ反転データ
- 24(AO).w 2 ページ反転データ
- 26(AO).w 3 ページ反転データ
- 28(AO).w コピー元の X オフセット
- 30(AO).w コピー元の Y オフセット
- 32(AO).1 コピー元の 1 ページのバイト数
- 34(AO).1 コピー先の 1 ページのバイト数
- 36(AO).1 「右から左コピー」フラグ

A1.1 コピー先のビットマップレコードへのポインタ

A2.1 コピー元のビットマップレコードへのポインタ

返り値 DO.1 リザルトコード

(3) hPutInit : 1 ラインの描画を行うための初期化ルーチン (1 ページのテキストタイプのみ)

引数 AO.1 水平ラインコピー用ワークへのポインタ

(AO).1	1	ラインコピーサブルーチンのアドレス
4(AO).1	0	ページコピールーチンエン트리アドレス
8(AO).1	1	ページコピールーチンエン트리アドレス
12(AO).1	2	ページコピールーチンエン트리アドレス
16(AO).1	3	ページコピールーチンエン트리アドレス
20(AO).w		フォアグラウンドカラー
22(AO).w		バックグラウンドカラー
24(AO).w	2	ページ反転データ
26(AO).w	3	ページ反転データ
28(AO).w		コピー元の X オフセット
30(AO).w		コピー元の Y オフセット
32(AO).1		コピー元の 1 ページのバイト数
34(AO).1		コピー先の 1 ページのバイト数
36(AO).1		「右から左コピー」フラグ

A1.1 描画を行うグラフポートレコードへのポインタ

返り値 DO.1 リザルトコード

●標準描画ルーチンのテーブル (\$A1E6 GMGetStdProcTbl)

グラフポートレコード中で指定される、グラフィックマネージャの描画ルーチンのテーブル (プロセスポインタ) と同形式です。

オフセット	欄 名	内 容	
+\$00.1	gText	文字列の描画ルーチン	(1)
+\$04.1	gLine	ラインの描画ルーチン	(2)
+\$08.1	gRect	レクタングルの描画ルーチン	(3)
+\$0c.1	gRRect	ラウンドレクタングルの描画ルーチン	(4)
+\$10.1	gOval	楕円の描画ルーチン	(5)
+\$14.1	gArc	円弧の描画ルーチン	(6)
+\$18.1	gPoly	ポリゴンの描画ルーチン	(7)
+\$1c.1	gRgn	リージョンの描画ルーチン	(8)
+\$20.1	gCopy	ビットイメージのコピールーチン	(9)
+\$24.1	gRsv1	システム予約	
+\$28.1	gRsv2	システム予約	
+\$2c.1	gWidth	文字列の幅を求めるルーチン	(10)
+\$30.1	gTinW	指定の幅に収まる文字列のバイト数を求めるルーチン	(11)
+\$34.1	gRsv3	システム予約	
+\$38.1	gRsv4	システム予約	
+\$3c.1	gRsv5	システム予約	

(1) gText: 文字列の描画ルーチン

引数 (AO).1 文字列へのポインタ

4(AO).1 文字列先頭からのオフセット

8(AO).w 文字列のバイト数
返回值 DO.1 リザルトコード

(2) gLine: ラインの描画ルーチン

引数 (AO).w 終点の x 座標
2(AO).w 終点の y 座標
返回值 DO.1 リザルトコード

(3) gRect: レクタングルの描画ルーチン

引数 (AO).w =0 枠を描画
=1 塗り潰して描画
2(AO).l レクタングルレコードへのポインタ
返回值 DO.1 リザルトコード

(4) gRRect: ラウンドレクタングルの描画ルーチン

引数 (AO).w =0 枠を描画
=1 塗り潰して描画
2(AO).l 外接するレクタングルレコードへのポインタ
6(AO).w x 方向の角の大きさ
8(AO).w y 方向の角の大きさ
返回值 DO.1 リザルトコード

(5) gOval: 楕円の描画ルーチン

引数 (AO).w =0 枠を描画
=1 塗り潰して描画
2(AO).l 外接するレクタングルレコードへのポインタ
返回值 DO.1 リザルトコード

(6) gArc: 円弧の描画ルーチン

引数 (AO).w =0 枠を描画
=1 塗り潰して描画
2(AO).l 外接するレクタングルレコードへのポインタ
6(AO).w 開始角度
8(AO).w 終了角度
返回值 DO.1 リザルトコード

(7) gPoly: ポリゴンの描画ルーチン

引数 (AO).w =0 枠を描画
 =1 塗り潰して描画
 2(AO).l ポリゴンレコードへのポインタ
 戻り値 DO.l リザルトコード

(8) gRgn: リージョンの描画ルーチン

引数 (AO).w =0 枠を描画
 =1 塗り潰して描画
 2(AO).l リージョンレコードへのハンドル
 戻り値 DO.l リザルトコード

(9) gCopy: ビットイメージのコピールーチン

引数 (AO).l コピー元ビットマップレコードへのポインタ
 4(AO).l コピー先ビットマップレコードへのポインタ
 8(AO).l コピー元レクタングルレコードへのポインタ
 \$c(AO).l コピー先レクタングルレコードへのポインタ
 \$10(AO).w コピーモード
 \$12(AO).l リスク範囲のリージョンレコードへのハンドル
 戻り値 DO.l リザルトコード

(10) gWidth: 文字列の幅を求めるルーチン

引数 (AO).l 文字列へのポインタ
 4(AO).l 文字列先頭からのオフセット
 8(AO).w 文字列のバイト数
 戻り値 DO.l 上位ワード, イタリックを含めた幅 (ドット数)
 下位ワード, イタリックを含めない幅 (ドット数)

(11) gTinW: 指定の幅に収まる文字列のバイト数を求めるルーチン

引数 (AO).l 文字列へのポインタ
 4(AO).l 文字列先頭からのオフセット
 8(AO).w ドット数
 戻り値 DO.l 文字列のバイト数

● 文字描画ルーチンのテーブル (\$A1E7 GMGetFontProcTbl)

オフセット	欄 名	内 容	
+\$00.l	faceJob	文字を修正するルーチン	(1)
+\$04.l	putCharJob	文字を描画するルーチン	(2)

(1) faceJob : 文字を修正するルーチン

引数 (SP).1 文字の描画されているビットマップレコードへのポインタ
 4(SP).1 文字の大きさを示すレクタングルレコードへのポインタ
 8(SP).1 フォントフェイス
 返回值 なし

フォントフェイスにしたがって文字を修正します。修正の結果、文字の大きさが変化した場合 (最大縦 4 ドット、横 4 ドットまで大きくなります) は、引数として渡したレクタングルレコードの内容を変化させます。イタリックの処理は行いません。

(2) putCharJob : 文字を描画するルーチン

引数 (SP).1 文字の描画されているビットマップレコードへのポインタ
 4(SP).1 文字の大きさを示すレクタングルレコードへのポインタ
 返回值 DO.l リザルトコード

カレントグラフポートに指定されたビットマップのレクタングルの範囲を文字として描画します。イタリックの処理はここで行われます。

● リージョン 1 行演算ルーチンのテーブル (\$A1E8 GMGetRgnProcTbl)

これらのルーチンは、リージョンの横 1 ラインを単位として処理を行います。したがって、引数となるのは、リージョンレコード中の横 1 ラインの先頭のアドレスです。

オフセット	欄 名	内 容	
+\$00.l	andRgnLine	リージョンの AND を求めるルーチン	(1)
+\$04.l	orRgnLine	リージョンの OR を求めるルーチン	(2)
+\$08.l	diffRgnLine	リージョンの差分を求めるルーチン	(3)
+\$0c.l	xorRgnLine	リージョンの XOR を求めるルーチン	(4)

(1) andRgnLine : リージョンの AND を求めるルーチン

引数 AO.l 結果を収めるバッファへのポインタ
 A1.l リージョン 1 へのポインタ
 A2.l リージョン 2 へのポインタ

返り値 AO.1 結果を収めるバッファへのポインタ (リージョン1 AND リージョン2)
 A1.1 リージョン1の次のラインへのポインタ
 A2.1 リージョン2の次のラインのポインタ
 DO-D3 不定

(2) orRgnLine: リージョンの OR を求めるルーチン

引数 AO.1 結果を収めるバッファへのポインタ
 A1.1 リージョン1へのポインタ
 A2.1 リージョン2へのポインタ
 返り値 AO.1 結果を収めるバッファへのポインタ (リージョン1 OR リージョン2)
 A1.1 リージョン1の次のラインへのポインタ
 A2.1 リージョン2の次のラインのポインタ
 DO-D3 不定

(3) diffRgnLine: リージョンの差を求めるルーチン

引数 AO.1 結果を収めるバッファへのポインタ
 A1.1 リージョン1へのポインタ
 A2.1 リージョン2へのポインタ
 返り値 AO.1 結果を収めるバッファへのポインタ (リージョン1 - リージョン2)
 A1.1 リージョン1の次のラインへのポインタ
 A2.1 リージョン2の次のラインのポインタ
 DO-D3 不定

(4) xorRgnLine: リージョンの XOR を求めるルーチン

引数 AO.1 結果を収めるバッファへのポインタ
 A1.1 リージョン1へのポインタ
 A2.1 リージョン2へのポインタ
 返り値 AO.1 結果を収めるバッファへのポインタ (リージョン1 XOR リージョン2)
 A1.1 リージョン1の次のラインへのポインタ
 A2.1 リージョン2の次のラインのポインタ
 DO-D3 不定

COLUMN TITLE.X の役割

システムディスクの中に収められている TITLE.X は、SX-WINDOW のオープニングタイトルを表示するだけのプログラムだと思われていますが、案外、いろいろな仕事をするプログラムです。

TITLE.X が使われるのは、①SX-WINDOW の起動時と、②SX-WINDOW の終了時で

す。いずれも、システムディスクのように SXWIN.X をシェルとして起動する場合なので、コマンドラインから SXWIN.X を起動している方はあまりお目にかかることはないかもしれません。

それぞれの場合の TITLE.X の動きについて説明します。

① SX-WINDOW の起動時

システムディスクの CONFIG.SYS の中で、TITLE.X に関係している部分を抜き出してみましよう。

```
PROGRAM =%SHELL%TITLE.X
```

TITLE.X は、この行によって起動され、下図のような画面を表示します。

■図 TITLE.X によるタイトル表示



このとき、SRAM が初期化されていなかった場合には初期化を行います。

また、TITLE.X の後ろにファイルネームを指定することによって、そのファイルにテキストタイプ2 ページのレクタングルイメージが収められていた場合、標準のタイトル画面のかわりに表示します。

② SX-WINDOW の終了時

SXWIN.X がシェルとして起動されている場合に、システムアイコンから「終了」が選択されると、シェルはリソース CMDS の 13 に収められているプログラムに、ShEV の 3 に収められているコマンドラインを渡して起動します（その際、タスクマネージャの FOCK 系ではなく、DOS コールの EXEC で起動します）。

標準の状態では、CMDS の 3 には TITLE.X が、ShEV の 3 には -e が収められているので、結局、TITLE.X が -e というオプション付きで起動されることになります。-e オプション付きで起動された場合、TITLE.X はいったんタイトル画面（この場合、つねに標準のタイトル画面で、イメージを収めたファイルを指定することはできません）を表示した後、徐々にフェードアウトし、もっとも暗くなったところで終了します。

TITLE.X（とはかぎりませんが）の処理が終わり、シェルに戻ると、シェルは無限ループに入り、電源オフを待つことになります。

2² | テキストマネージャ

SX1.02のテキストマネージャは、いちおうの文書編集機能を有していたとはいえ、スピード的な問題や機能的な問題を抱えており、事実上、ファイル名の入力程度にしか使われていませんでした。新しいテキストマネージャでは、これらの問題はクリアされており、実用に耐えるものとなっています。標準添付のエディタ.Xがその改良の結果を如実に物語っています*4。

*4:本書も一部エディタ.Xによって執筆されています。

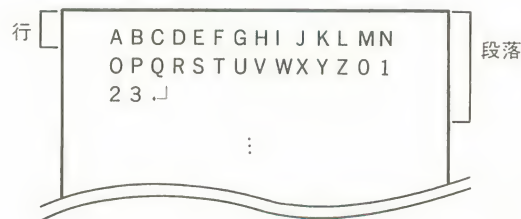
改良されたとはいえ、SX1.02との上位互換性は保たれているので、これまでの利用方法をそのまま適用することができます。エディタやワープロなど高度なアプリケーションを作成される方以外は、とくにプログラミングスタイルを変更することなく、スピードアップの恩恵のみを被ることができます。

ここでは、おもな変更点について解説することになります。

1 | 行と段落の概念

新しいテキストマネージャでは、画面上で横1行に表示される文字列を「行」、メモリ内の行末コード(\$OD\$OA)までの文字列を「段落」と呼びます。前者をいわゆる「物理行」、後者を「論理行」といい替えることもできます(図5)。

■図5 行と段落の関係



一見同じもののようにも思えますが、行と段落が違うものになるのは、ディスプレイネーションレクタングルの横方向のサイズによって、行の折り返しが発生するからです。このあたりについては、前著『SX-WINDOW〜』で解説済みですので、繰り返しません。

2 | テキストエディットレコードの変更

各種の拡張に対応するため、テキストエディットの内容がかなり変更されています。基本的にSX1.10は、それ以前のシステムとの上位互換性が維持されていますが、テキストエディットレコードに関して完全に互換というわけにはいかなかったようです。レコードの内容を直接

参照/変更しているアプリケーションは注意が必要です。

新しいテキストエディットレコードの内容は次のとおりです。SX1.10以降で新しく用意された欄のうち、必要と思われるものについてのみ解説を行うことにします。

オフセット	欄 名	内 容	
+\$00	teDestRect	ディスティネーションレクタングル	
+\$08	teViewRect	ビューレクタングル	
+\$10.l	teDestOffsetX	水平方向補正座標	(1)
+\$14.l	teDestOffsetY	垂直方向補正座標	(2)
+\$18.l	teHText	編集テキストへのハンドル	(3)
+\$1c.l	teLenMax	編集テキストの最大サイズ	
+\$20.l	teLength	すでに入力されている編集テキストのサイズ	
+\$24.l	teRsv0	システム予約	
+\$28.l	teSelStart	セレクト範囲の開始位置	
+\$2c.l	teSelEnd	セレクト範囲の終了位置	
+\$30.l	teSelLine	現在のセレクト行位置	
+\$34.l	teSelOffset	現在のバッファのセレクト位置	
+\$38.l	teRefCon	ユーザ定義データ	(4)
+\$3c.w	teLineHeight	改行幅 (ドット単位)	
+\$3e.w	teTabSize	TAB サイズ (ドット単位)	
+\$40.w	teJust	行揃えモード (0:左寄せ, 1:中央寄せ, -1:右寄せ)	
+\$42.b	teDrawMode	編集モード	(5)
+\$43.b	teVis	ドローレベル	(6)
+\$44.l	teSelLocateX	水平座標ロング値	(7)
+\$48.l	teSelLocateY	垂直座標ロング値	(8)
+\$4c.l	teInPort	グラフポートレコードへのポインタ	(9)
+\$50.l	teCaretTime	内部で使用 (キャレットの点滅間隔)	
+\$54.w	teCaretState	内部で使用 (キャレットの状態/ハイライト表示レベル)	(10)
+\$56.l	teFunction	プロセステーブルへのポインタ	(11)
+\$5a.l	teFuncCode	ファンクションキーアサインテーブルへのポインタ	(12)
+\$5e.l	teCtrlCode	コントロールキーテーブルへのポインタ	(13)
+\$62.l	teCtrlFun	コントロールキー処理ルーチンテーブルへのポインタ	(14)
+\$66.l	teNColumns	テキストの段落数	
+\$6a.l	teNLines	テキストの行数	
+\$6e.l	teLineStarts	内部で使用 (行頭テーブルへのハンドル)	(15)

(1) 水平方向補正座標

(2) 垂直方向補正座標

ビューレクタングルに対するディスティネーションレクタングルのオフセットが入ります。

(3) 編集テキストへのハンドル

編集テキストが収められている再配置可能ブロックへのハンドルが収められています。

テキストが収められるブロックは新しくテキストエディットを作成する際に作成されますが、アプリケーション側で用意したものをセットすることも可能です。その際、テキストのサイズよりも最低 8 バイト以上大きなブロックを用意するようにしてください。

(4) ユーザ定義データ

ユーザが自由に使用することができます。

(5) 編集モード

テキストエディットに関する各種フラグの集合体です。各ビットは、次のような意味を持っています。

意 味		値	1	0
bit0	改行コードの表示		表示する	表示しない
bit1	テキスト終端の表示		表示する	表示しない
bit2	コントロールコードの表示形式		$\wedge A$	S_B
bit3	コントロールコードの編集許可		許可する	許可しない
bit4	リードオンリー属性		リードオンリー	編集可
bit5	下揃え		下揃えする	下揃えしない

(6) ドローレベル

ドローレベルが 0 以上の場合にテキストエディットの描画を行い、負の数の場合は描画を行いません。ドローレベルは描画だけを制御するもので、描画が行われていない場合でも、編集の処理は通常どおりに行われます。

(7) 水平座標ロング値

(8) 垂直座標ロング値

ディスティネーションレクタングル内部でのカーソルの位置をロングワードで表現します。

(9) グラフポートレコードへのポインタ

SX1.02 までは、テキストエディットの表示を行いたいグラフポートを、表示の前にカレントにしておく必要がありましたが、SX1.10 からはテキストエディットレコードの中に表示を行うグラフポートへのポインタを収めるようになりました。これによって、カレントグラフポートをセットする必要はなくなりました。

このグラフポートに設定されているフォント等を変更することにより、描画するフォント等も変わるわけですが、その場合には \$A464 TMSetSelGal を呼んで、テキストエディットレコード全体を再計算させたくて再描画するようにしてください。

(10) ハイライト表示レベル

ハイライトレベルが 0 以上の場合、セレクト範囲のハイライト表示を行い、負の数の場合は描画を行いません。

(11) プロセステーブルへのポインタ

テキストの編集や表示を行うためのルーチンはプロセステーブルに登録されており、テキストマネージャはここに収められているプロセステーブルへのポインタを参照して各処理へ分岐します。したがって、ここに別のルーチンを登録したプロセステーブルへのポインタを格納しておくことによって、これらのルーチンを差し替えることが可能です。

デフォルトでは SX-SYSTEM 内のテーブルを指しているため、直接テーブルを書き換えようすると、バスエラーが発生する場合があります。あらかじめ、ほかの場所に標準のテーブルをコピーしておいて、そこを書き換えたうえでテキストエディットレコードに登録してく

ださい。

詳しい内容については後述します。

(12) ファンクションキーアサインテーブルへのポインタ

定義可能キー (F1~F20, カーソルキー, ROLL UP/DOWN, UNDO 等) にコントロールコードを割り当てるのが, ファンクションキーアサインテーブルの役割です。

ファンクションキーアサインテーブルの形式を, 次の例で示します。

funcAssignTable :		* キーコード, ASCII コード
dc.w	\$003C,\$10	* カーソル上, \$10 (^P)
dc.w	\$003E,\$0E	* カーソル下, \$0E (^N)
	:	
dc.w	0	* エンドマーク

(ラベルは筆者が独自に定義したものです)

1 ワードずつ, キーコード (IOCS の `_B_KEYINP`, `_B_KEYSNS` 等で使われる内部コード) と, そのキーに対応させたいコントロールコードを並べ, 最後にエンドマークとして \$00 を置きます。

上の例では, 編集中にカーソルキーの上が押された場合は ^P が押されたのと, カーソルキーの下が押された場合は ^N が押されたのと同じことになります。

デフォルトでは, SX-SYSTEM 内のテーブルを指しているので, 直接テーブルを書き換えようとする, バスエラーが発生する場合があります。あらかじめ, ほかの場所に標準のテーブルをコピーしておいて, そこを書き換えたいのでテキストエディットレコードに登録してください。

(13) コントロールキーテーブルへのポインタ

(14) コントロールキー処理ルーチンテーブルへのポインタ

編集中にコントロールコードが入力された場合の処理を登録しておくテーブルがコントロールキー処理ルーチンテーブルです。

SX1.02 のテキストエディットにも同様なテーブルが用意されていましたが, 多少形式が変更されているので注意が必要です。

コントロールキーテーブルの形式は SX1.02 から変わっていません。次の例のような形式です。

ctrlKeyTable :		
dc.w	\$0d	* リターンキー
dc.w	\$08	* ^H (BS)
	:	


```
dc.w      0      *エンドマーク
```

(ラベルは筆者が独自に定義したものです)

これに対応するルーチンを列挙するコントロールキー処理ルーチンテーブルは、次のような形式となります。

```
ctrlKeyProc:
    dc.l      CRproc      *リターンキーを押した処理
    dc.l      BSproc      *BS キーを押した処理
    :
```

(ラベルは筆者が独自に定義したものです)

SX1.02 では処理ルーチンへのオフセットで表現していましたが、新しいテキストマネージャではポインタとなっています。

コントロールキーテーブル、コントロールキー処理ルーチンテーブルとともに、デフォルトでは SX-SYSTEM 内のテーブルを指しているため、直接テーブルを書き換えようとするとバスエラーが発生する場合があります。あらかじめ、ほかの場所に標準のテーブルをコピーしておいて、そこを書き換えたいうえでテキストエディットレコードに登録してください。

(15) 行頭テーブルへのハンドル

行頭テーブルは、SX1.02 にくらべ大幅に構造が変更されています。行頭のオフセットを単純に並べたものではなく、各行のバイト数や行の状態などの情報が格納されるテーブルになりました。

各行（段落）の状態を得るための SX コールがいくつか用意されたので、アプリケーションが直接参照/変更するメリットはもはやないと思われます。

3 段落情報

従来は、ある行がテキスト全体の中でどのような位置にあるのか、などといった情報を得る手段がありませんでした。また、行単位、段落単位での現在位置の把握も不可能でした。こうした情報は、本格的なエディタやワープロをつくるためにはどうしても必要なものです。

新しいテキストマネージャには、こうした情報を得るための SX コールが何種類か用意されていますが、これらはいずれも情報を段落情報というかたちで返してきます。

段落情報レコードの形式は次のとおりです。

オフセット	欄 名	内 容
+\$00.l	teCoPos	段落位置
+\$04.l	toCoCnt	段落の行数
+\$08.l	teCoLine	段落の先頭行位置
+\$0c.l	teCoOffset	段落の先頭行のオフセット
+\$10.l	teCoSize	段落のバイト数
+\$14.l	teCoPtr	段落へのポインタ
+\$18.l	teLine	行位置
+\$1c.l	teLOffset	行の先頭位置オフセット
+\$20.l	teLSize	行のバイト数
+\$24.l	teLPtr	行へのポインタ

4

編集履歴

SX1.02 までは、編集を行った結果、再描画が必要となった場合にはビューレクタングル全体を書き換えていました。これが処理スピードを低下させる原因の1つでもありました。これを避けるためには、変更があった場所のみを書き換えればよいことはすぐに気がつきます。

編集履歴は、どこで、どのような変更があったのかを記録しておくためのデータ構造です。その内容は編集履歴レコードにまとめられます。編集履歴レコードの形式は以下のとおりです。

オフセット	欄 名	内 容
+\$00.w	teEditOn	編集されたかどうかのフラグ (≠0: 編集された)
+\$02.l	teEditnLine	編集行数
+\$06.l	teEditStart	編集開始バイト位置
+\$0a.l	teEditLine	編集開始行位置
+\$0e.l	teEditLocateH	編集開始水平座標
+\$12.l	teEditLocateV	編集開始垂直座標
+\$16.l	teEditLocateM	編集開始座標修正値
+\$1a.l	teEditCoLine	次の段落の先頭行位置
+\$1e.l	teEditCodiff	次の段落がスクロールしたかどうかのフラグ (≠0: スクロールした)

編集履歴は\$A348 TMStr2 によって作成され、\$A338 TMUpDate2 によって解釈され、描画が行われます。

5

キャッシュ機能の追加

スピード向上のもう1つの方策として、キャッシュ機能の追加があります。キャッシュにはテキストの一部が読み込まれ、編集作業は一時的にこの中で行われます。

大きなテキストの場合、細かい変更を行うたびにテキスト全体を操作していたのでは時間がかかりすぎます。キャッシュを追加したことにより、編集の結果を効率よくテキスト全体に反映させることができるようになっていきます。

キャッシュが追加されたことにより、2つほど注意が必要になっています。

(1) テキストを直接参照する場合

テキストを直接参照する場合は、変更した結果がキャッシュに残っていて、テキストに反映されていない場合がありますから、必ずキャッシュをフラッシュしてから参照するようにしてください。

(2) \$A33C TMCaLine で段落を参照する場合

指定した行またはバイト位置の段落情報を作成する\$A33C TMCaLineでは、段落情報の中に段落へのポインタを返します。これを利用してテキストを直接参照することも可能ですが、この段落がキャッシュ中に残っている場合、キャッシュ中のアドレスが返される場合があります。このアドレスを使って、この段落に正しく参照することができますが、このほかの段落については保証のかぎりではありません。

キャッシュを操作するSXコールとして、\$A32C TMCacheON、\$A32D TMCacheOFF、\$A32ETM CacheFlushが用意されています。

6**アップデート処理の充実**

編集履歴によって部分的な書き換えがサポートされたことなどから、多彩なアップデートが用意されています。

このうち、SXコールでサポートされるものが3種類、ライブラリでのみサポートされるものが2種類あります。それぞれについて解説します。

<SXコールでサポートされるもの>**●\$A313 TMUpdate**

これはSX1.02から用意されているSXコールです。指定したレクタングルの内部を背景色で塗り潰してから再描画します。スピード的にはもっとも不利ですが、従来の手順を変更してまでほかのSXコールを利用するかどうか、適切に判断して利用してください。

●\$A338 TMUpdate2

\$A348 TMStr2によって文字列を挿入した後でのみ利用可能です。TMStr2が作成した編集履歴を参照して描画を行います。条件は限定されますが、これによってアップデートを行うのがもっとも速く行えます。

●\$A339 TMUpdate3

指定したレクタングルの内部を再描画するのは\$A313 TMUpdateと同じですが、背景色での塗り潰しを行わない点で異なっています。塗り潰しを行わない分、TMUpdateよりは高速です。

<ライブラリでのみサポートされるもの>

●TMEventW

\$A31C TMEvent と同様な処理を行います³、その前にアップデートリージョンへの再描画を行います。テキストエディットレコードに収められているグラフポートがウィンドウレコードの一部であると仮定して処理を行います。

具体的には、アップデートリージョンとクリップリージョンが重なった範囲を再描画し、アップデートリージョンから除いた後で、TMEvent を呼んでいます。

●TMUpdateExist

アップデートリージョンとクリップリージョンが重なった範囲を再描画し、指定によってはその範囲をアップデートリージョンから除きます。テキストエディットレコードに収められているグラフポートがウィンドウレコードの一部であると仮定して処理を行います。

編集を行ったときに、スクロールが発生する場合があります³、このときにアップデートされていない領域が残っていると、その後、正常にアップデートできなくなります。そのため、スクロールが発生する可能性のある編集を行う場合は、あらかじめ、この TMUpdateExist を呼び出して、アップデートされていない部分を再描画しておくようにしてください。

7

プロセステーブルの拡張

プロセステーブルの形式は以下のとおりです。

オフセット	欄 名	内 容	
+\$00.l	OFF_InWid	指定した幅に収まる文字数を求めるルーチンのアドレス	(1)
+\$04.l	OFF_Width	文字列の幅を求めるルーチンのアドレス	(2)
+\$08.l	OFF_Draw	文字列の描画ルーチンのアドレス	(3)
+\$0c.l	OFF_UpDt	指定された範囲の描画ルーチンのアドレス (背景色での塗り潰し付き)	(4)
+\$10.l	OFF_UpDt2	編集履歴による再描画ルーチンのアドレス	(5)
+\$14.l	OFF_UpDt3	指定された範囲の描画ルーチンのアドレス (背景色での塗り潰しなし)	(6)
+\$18.l	OFF_Rev	セレクト部分のハイライト表示ルーチンのアドレス	(7)
+\$1c.l	OFF_Scroll	テキストエディットレコードのスクロール処理ルーチンのアドレス	(8)
+\$20.l	OFF_ScrollR	指定された範囲のスクロール処理ルーチンのアドレス	(9)
+\$24.l	OFF_FillR	指定された範囲の塗り潰しルーチンのアドレス	(10)
+\$28.l	OFF_CLIP	マウスによるセレクト処理ルーチンのアドレス	(11)
+\$2c.l	OFF_Caret	キャレット描画ルーチンのアドレス	(12)
+\$30.l	OFF_DrEOF	テキスト終端の描画ルーチンのアドレス	(13)
+\$34.l	OFF_Str	文字列挿入ルーチン (表示はしない) のアドレス	(14)
+\$38.l	OFF_Sel	セレクト部分の変更ルーチンのアドレス	(15)
+\$3c.l	OFF_FillRL	1行単位の塗り潰しルーチン(イタリック対応)のアドレス	(16)
+\$40		システム予約 (4 ロングワード)	

それぞれのルーチンが満たすべき仕様だけを示すことにします。

(1) OFF_InWid : 指定した幅に収まる文字数を求めるルーチン

パラメータ	(AO).l	テキストエディットレコードへのハンドル
	4(AO).l	文字列へのポインタ
	8(AO).l	文字列の先頭からのオフセット
	\$c(AO).w	文字列を収める幅 (ドット数)
	\$e(AO).w	文字列のローカル座標-デイスティネーションレクタングルのオフセット値
返り値	DO.l	文字列のバイト数
	AO.l ≠ 0	改行コードにより終了した
	= 0	それ以外

指定された幅に収まる文字列のバイト数を返します。

(2) OFF_width : 文字列の幅を求めるルーチン

パラメータ	(AO).l	テキストエディットレコードへのハンドル
	4(AO).l	文字列へのポインタ
	8(AO).l	文字列の先頭からのオフセット
	\$c(AO).w	文字列のバイト数
	\$e(AO).w	文字列のローカル座標-デイスティネーションレクタングルのオフセット値
返り値	DO.l	文字列の幅 (ドット数)

指定した文字列が占める幅を返します。

(3) OFF_Draw : 文字列の描画ルーチン

パラメータ	(AO).l	テキストエディットレコードへのハンドル
	4(AO).l	文字列へのポインタ
	8(AO).l	文字列の先頭からのオフセット
	\$c(AO).w	文字列のバイト数
	\$e(AO).w	文字列のローカル座標-デイスティネーションレクタングルのオフセット値
	\$lO(AO).w	0 以外を指定すると、タブ描画時に背景色で塗り潰しを行う
返り値	DO.l	リザルトコード

指定した文字列を描画します。クリップリージョンは設定されています。teDrawModeを参照して、モードにあわせた描画を行ってください。

(4) OFF_UpDt: 指定された範囲の描画ルーチン (背景色での塗り潰し付き)

パラメータ (AO).1 テキストエディットレコードへのハンドル
 4(AO).1 再表示する範囲を意味するレクタングルレコードへのポインタ
 戻り値 DO.1 リザルトコード

指定された範囲を背景色で塗り潰してから再描画します。以下の条件のもとで処理を行ってください。

- ・ドローレベルが負の場合は描画を行わない
- ・カーソルを消す
- ・ビューレクタングル、クリップリージョンの重なりを計算する
- ・ハイライト表示属性がオンの場合、セレクト部分のハイライト処理ルーチンを呼び出してハイライト表示を行う
- ・teDrawMode を参照して、モードにあわせた描画を行う

(5) OFF_UpDt2: 編集履歴による再描画ルーチン

パラメータ (AO).1 テキストエディットレコードへのハンドル
 4(AO).1 編集履歴レコードへのポインタ
 戻り値 DO.1 リザルトコード

編集履歴にしたがって描画を行います。OFF_UpDt と同様な条件のもとで描画を行ってください。

(6) OFF_UpDt3: 指定された範囲の描画ルーチン (背景色での塗り潰しなし)

パラメータ (AO).1 テキストエディットレコードへのハンドル
 4(AO).1 再表示する範囲を意味するレクタングルレコードへのポインタ
 戻り値 DO.1 リザルトコード

指定された範囲を背景色で塗り潰さずに再描画します。OFF_UpDt と同様な条件のもとで描画を行ってください。

(7) OFF_Rev: セレクト部分のハイライト表示ルーチン

パラメータ (AO).1 テキストエディットレコードへのハンドル
 4(AO).1 開始位置
 8(AO).1 終了位置
 戻り値 DO.1 リザルトコード

開始位置から終了位置までをハイライト表示します。ビューレクタングル、クリップリージョンの重なりを計算する必要があります。ドローレベル、ハイライト表示レベルのどちらかが負の場合は描画する必要はありません。

(8) OFF_Scroll : テキストエディットレコードのスクロール処理ルーチン

パラメータ (AO).1 テキストエディットレコードへのハンドル
 4(AO).1 水平方向ドット数
 8(AO).1 垂直方向ドット数
 戻り値 DO.1 リザルトコード

指定したドット数だけ縦横にスクロールさせます。次の条件のもとで処理を行ってください。

- ・ドローレベルが負の場合は描画を行わない
- ・カーソルを消す
- ・ビューレクタングル、クリップリージョンの重なりを計算する
- ・水平方向補正座標、垂直方向補正座標、水平座標ロング値、垂直座標ロング値を更新する
- ・実際のスクロールは OFF_ScrollR を呼び出して行う
- ・スクロールの結果、再表示が必要な場合は描画が必要

(9) OFF_ScrollR : 指定された範囲のスクロール処理ルーチン

パラメータ (AO).1 テキストエディットレコードへのハンドル
 4(AO).1 スクロールを行う範囲を意味するレクタングルレコードへのポインタ
 8(AO).1 スクロールオフセットを示すポイント
 \$c(AO).1 再表示が必要な範囲を意味するレクタングルレコードへのポインタ
 戻り値 DO.1 リザルトコード
 AO.1 リージョンレコードへのハンドル

指定したレクタングルの範囲内を、指定したドット数だけ縦横にスクロールさせます。スクロールによって生じた空白部分を背景色で塗り潰し、その範囲をリージョンに格納し、AO に返します。

(10) OFF_FillR : 指定された範囲の塗り潰しルーチン

パラメータ (AO).1 テキストエディットレコードへのハンドル
 4(AO).1 塗り潰しを行う範囲を意味するレクタングルレコードへのポインタ
 戻り値 DO.1 リザルトコード

指定された範囲内を背景色で塗り潰します。

(11) OFF_CLIP : マウスによるセレクト処理ルーチン

パラメータ (AO).1 テキストエディットレコードへのハンドル
 4(AO).1 マウス座標を意味するポイント
 8(AO).w 0 : 新規
 1 : 前回の選択位置を変更

2: 開始位置を変更

3: 終了位置を変更

戻り値 DO.1 リザルトコード

マウスによってセレクト位置を変更/設定します。デフォルトでは、座標をロングワードに変換して、\$A33F TPointSel を呼んでいるだけです。

(12) OFF_Caret: カーソル描画ルーチン

パラメータ (AO).1 テキストエディットレコードへのハンドル

戻り値 DO.1 リザルトコード

セレクト位置にカーソルを表示します。ペンモードは XOR に設定されているので、同じ図形を描画することで表示/消去が行われます。クリップリージョンは設定済みです。

(13) OFF_DrEOF: テキスト終端の描画ルーチン

パラメータ (AO).1 テキストエディットレコードへのハンドル

戻り値 DO.1 リザルトコード

テキスト終端記号を描画します。終端位置にペン位置がセットされています。

(14) OFF_Str: 文字列挿入ルーチン (表示はしない)

パラメータ (AO).1 テキストエディットレコードへのハンドル

4(AO).1 文字列へのポインタ

8(AO).1 文字列のバイト数

\$c(AO).1 編集履歴レコードへのポインタ

戻り値 DO.1 =0 選択部分も挿入文字列もない

=1 編集した

=-10240 最大入力数を超える

=-10239 リードオンリーモード

セレクト範囲と文字列を交換し、編集履歴を記録します。カーソル位置の再計算を行う必要はありません。キャッシュやテキストのメモリブロックを作成したり、行頭テーブルの設定を行う等、複雑な処理が要求されます。

(15) OFF_Sel: セレクト部分の変更ルーチン

パラメータ (AO).1 テキストエディットレコードへのハンドル

4(AO).1 新しいセレクト範囲の開始位置

8(AO).1 新しいセレクト位置の終了位置

\$c(AO).1 現在の新しいセレクト位置

戻り値 DO.1 リザルトコード

セレクト位置を指定の位置に変更します。新しいセレクト範囲と、それまでのセレクト範囲の重なりを調べ、OFF_Rev を呼び出して変更部分だけ反転させます。テキストのバイト数を超えた値の場合は補正する必要があります。カーソル位置の再計算を行う必要はありません。

(16) OFF_FillRL : 1 行単位の塗り潰しルーチン (イタリック対応)

パラメータ (AO).l テキストエディットレコードへのハンドル
 4(AO).l 塗り潰す範囲を示すレクタングルレコードへのポインタ
 返り値 DO.L リザルトコード

指定された範囲内を背景色で塗り潰します。フォントフェイスにイタリックが指定されている場合は、指定されたレクタングルをイタリックと同様に傾けて塗り潰しを行います。

8

そのほか

エディタ等の文書を編集する本格的なアプリケーションを支援するために、さまざまなユーティリティ的な機能がテキストマネージャに追加されています。代表的なものを列挙してみます。

- ・ テキスト中の文字列の前/後方検索
- ・ キー入力関係のユーティリティ (キーコードの変換, キーの先読み等)
- ・ etc.

このほかに、些末なことですが、SX1.10 からテキストエディット中でポップアップするメニューがリソース MENU の -4096 として定義されています。このリソースは SYSTEM.LB に収められており、\$A30A TMInit 内で読み込まれます。リソースを差し替えることによって内容を変更することは可能ですが、カット、コピー等のアイテム番号は固定ですから、表記を変える程度の変更しかできないと思われます。

COLUMN HENWIN.X の役割と動作の仕組

SX-WINDOW ユーザーズマニュアルでは、HENWIN.X は「漢字変換ウィンドウです」と説明されていますが、もう少し補足説明を加えておきたいと思います。

HENWIN.X は、ユーザが直接実行するためのファイルではありません。シェルが立ち上がった時点で自動的に起動し、シェルが終了するまで1つのタスクとして動き続けます。このタスクの役割はというと、キーボードから文字を入力中にフロントプロセッサを起動した場合に、画面下部に変換ウィンドウを表示し、変換過程を表示することにあります。この仕事かどのようなものであるかは、HENWIN.X をほかのファイル名にリネームしたうえでシェルの起動して、フロントプロセッサを起動してみた状態と比較してみればよくわかると思い

ます。

前著『SX-WINDOW〜』では、ウィンドウ ID\$28 のウィンドウを「漢字変換用」としてしまいましたが、実際の変換ウィンドウはウィンドウマネージャを使わず、グラフィックマネージャを用いて直接描画した特殊なウィンドウとなっています。

この動作は、DOS コールの\$FF18 hendsp をトラップすることによって実現されています。hendsp は、フロントプロセッサのための DOS コールで、システム行に漢字変換過程や候補などを表示する機能を持っています。Human の用意する標準の hendsp のルーチンでは、つねに画面最下行に変換過程を表示するようになっていますが、これを SX-SYSTEM 内のルーチンと差し替えることによって、SX-WINDOW のデスクトップ上のウィンドウ風の領域の中に表示させているわけです。

2³ タスクマネージャ

SX-WINDOW で動作するすべてのタスクを統括するタスクマネージャの基本的な機能にはなんら変更はありません。バージョンアップした点としては、いくつかの小規模な変更と、ユーティリティ的 SX コールの追加が挙げられます。

1 モジュールヘッダの拡張

SX1.10 では、モジュールヘッダの内容が拡張され、次のような形式になっています。

オフセット	内 容	
+\$00.1	モジュールタイプ	(1)
+\$04.1	モジュールサイズ	
+\$08.1	スタートアドレスオフセット	
+\$0c.1	ワークサイズ	
+\$10.1	コモンエリアサイズ	(2)
+\$14.1		
⋮	システム予約	
+\$1c.1		

(1) モジュールタイプ

従来のモジュールタイプは OBJR, OBJC, OBJO の 3 種類でしたが、OBJX という新しいモジュールタイプが追加されています*5。

*5:標準のシステムでは、HD フォーマット X が X 型のモジュールです。

X 型のモジュールは単独実行型と呼ばれ、シェル上で動作しません。実行が開始されるのは、つねにコマンドラインから起動された場合のエントリポイント(アセンブラソースで .end 命令によって指定したスタートアドレス) からです。必要な場合は、ここで外部カーネル等呼び出して SX-SYSTEM を初期化してください。

シェルから起動した場合、Human のプログラムを実行したときと同様、その時点で走行していたほかのタスクには、タスクマネージャイベントの 31(SAVE), 33(NOTICEENDTSK), 1(ENDTSK) が送られて実行の終了が要請され、すべてのタスクが終了した段階で X 型のモジュールが実行されます。Human のプログラムと異なり、モジュールの動作が終了した際に「シェルに戻ります何かキーを押してください!!」のメッセージが表示されず、キーが押されるのを待たずにシェルに復帰します。

(2) コモンエリアサイズ

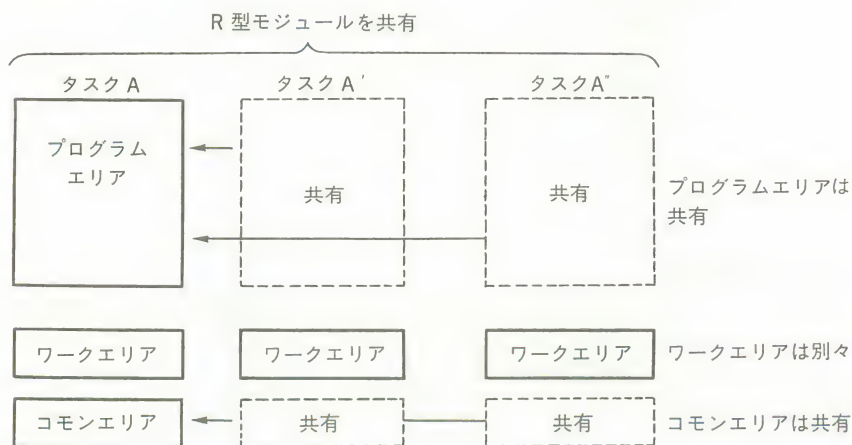
R 型のモジュールの場合、プログラムエリアはいくつかのタスクで共有されます。プログラムエリアが「プログラム」+「固定データ」+「静的なワーク」で成り立っていることは、前著『SX-WINDOW〜』で説明しましたが、このうちの「静的なワーク」も共有されるこ

とはいうまでもありません。これをうまく使えば、プログラムエリアを共有しているタスク間でかんたんに情報を共有し、また、やりとりすることが可能となります。

これを明示的に行うのがコモンエリアです。コモンエリアは、プログラムエリアを共有しているタスクの数に関わらず、1つのモジュールにつき1つ用意されます*6。プログラムエリアを共有するタスクには、同じコモンエリアの先頭アドレスが通知され、同じ情報へアクセスすることが可能です (図6)。

*6: コモンエリアを利用しているアプリケーションとしては、エディタ.Xが挙げられます。エディタ.Xでは、複数のファイルを編集しているとき、[ESC] + [E] ですべてのファイルの編集 (=複数のエディタ.X) が終了しますが、これはコモンエリアによって連絡をとりあっていると思われます。

■図6 コモンエリアの概念



X タイプの実行ファイルの場合は、静的なワークを BSS セクションに置くことでファイルサイズを節約できますが、R タイプの実行ファイルでは BSS 領域が用意されないので、この方法が使えません。このような場合、コモンエリアを利用することで BSS 領域のかわりとなることが出来ます。

コモンエリアが必要ない場合は、この欄には 0 を入れておきます。

2 起動時のレジスタ内容の変更

コモンエリアを利用するためには、この先頭アドレスがアプリケーションに通知されなければなりません。SX1.10 では、タスクが起動された際にレジスタ A4 によって、この値が渡されます。A1 に収められているワークエリアの先頭アドレスを利用するときと同様に、A4 を利用してコモンエリアにアクセスするようにしてください。

この結果、タスクの起動時には各レジスタに次のような値が入っていることになります。

レジスタ	内 容
D0	自分に与えられたタスク ID
D1	コードを共有するタスクの ID
A1	ワークエリアの先頭アドレス
A2	コマンドラインのアドレス
A3	環境のアドレス
A4	コモンエリアの先頭アドレス*
USP	スタックトップ (ワークエリアの最後+1)

(*が追加/変更された欄)

コモンエリアが必要ないモジュールの場合は、A4 には 0 が入ります。

3 タスク間通信の手順の変更

SX1.02 では、\$A35F TSCCommunicate の不備 (3 者以上でタスク間通信を行った場合の混乱等) を回避するために、TSSendMes, TSAnswer をライブラリとして用意し、それを使うことが推奨されていました。SX1.10 では、これらは SX-SYSTEM 内に統合されています。これにより、タスク間通信は、\$A417 TSAnswer, \$A418 TSSendMes, \$A419 TSGetMes を利用して行うことになります。

4 タスク管理テーブルの拡張

タスク間通信が改良/変更、コモンエリアの追加により、タスク管理テーブルにそれらの情報を格納するための欄が追加されています。この結果、タスク管理テーブルの形式は次のようになりました。

オフセット	欄 名	内 容
+\$000	tsName	タスク名 (ASCIIZ)
+\$05a	tsCommand	コマンドライン (LASCII)・
+\$15a.w	tsTskID	自分のタスク ID
+\$15c.w	tsParentID	親のタスク ID
+\$15e.w	tsStMode	立ち上げモード
+\$160.l	tsRscType	リソースタイプ
+\$164.w	tsRscID	リソース ID
+\$166.w	tsState	現在の状態
+\$168.l	tsProgramPtr	プログラムエリアへのポインタ
+\$16c.l	tsProgramHdl	プログラムエリアへのハンドル
+\$170.l	tsDaraHdl	ワークエリアへのハンドル
+\$174.l	tsEnvPtr	環境エリアへのポインタ
+\$178.l	tsD1RegKeep	D1
+\$17c.l	tsD2RegKeep	D2
+\$180.l	tsD3RegKeep	D3
+\$184.l	tsD4RegKeep	D4
+\$188.l	tsD5RegKeep	D5
+\$18c.l	tsD6RegKeep	D6
+\$190.l	tsD7RegKeep	D7

タスク切り替え時の
レジスタ退避用

(前ページ続き)

オフセット	欄 名	内 容
+\$194.l	tsA1RegKeep	A1
+\$198.l	tsA2RegKeep	A2
+\$19c.l	tsA3RegKeep	A3
+\$1a0.l	tsA4RegKeep	A4
+\$1a4.l	tsA5RegKeep	A5
+\$1a8.l	tsA6RegKeep	A6
+\$1ac.l	tsA0RegKeep	A0
+\$1b0.l	tsD0RegKeep	D0
+\$1b4.w	tsSrRegKeep	SR
+\$1b6.w		システム予約
+\$1b8.l	tsSpRegKeep	USP
+\$1bc.l	tsPcRegKeep	PC
+\$1c0.w	tsCommSendID	最後に送信したタスクの ID*
+\$1c2.w	tsCommRecvID	現在受けているタスク間通信の 送信元タスクの ID*
+\$1c4.l	tsTickCount	起動時のシステム時刻*
+\$1c8.l	tsCommonHdl	コモンエリアへのハンドル*
+\$1cc		
⋮	tsRsv	システム予約
+\$1ff		

(*が追加/変更された欄)

5 タスクマネージャイベントの拡張

システム全体をより効率よく、安全に使用するために、タスクマネージャから各タスクへシステムの状況等を通知するためのメッセージであるタスクマネージャイベントにいくつかの新しいイベントが追加されました。

追加されたタスクマネージャイベントについて次ページの上表に示します。

デスクトップスクラップやドラッグのデータをやりとりするためのセルレコードで、画像関係の情報を扱うことができるようになりました。

6 セルレコードに登録されるデータの種類の追加

従来から扱うことができた F\$?? (アイコン情報)、STRN (文字列情報) とともに、次ページの下表に示す情報が扱えるようになっています。従来からの情報とあわせて示します。

タスクマネージャイベントコード (略称)	イベントの内容	
	引数 1	引数 2
33 (NOTICEENDTSK)	全タスク終了の予告です。このイベントが届いた時点で終了すると都合が悪いタスクは、\$ A358 TSGetEvent でこのイベントを取り除いてください。	
	0 : X 型モジュールを実行 1 : 再起動 2 : 終了	0
70 (CREATETSK)	タスクの起動を通知します。	
	タスク ID	下記のデータへのハンドル +\$000 タスク管理テーブルのコピー
71 (EXITTSK)	タスクの終了を通知します。	
	タスク ID	下記のデータへのハンドル +\$000 タスク管理テーブルのコピー +\$200.1 終了コード
91 (DELETETSK)	リソースの削除を通知します。	
	リソースのタイプ	リソースへのハンドル

情報の種類	情報の内容	格納形式
FS?? (??は英字 2 文字)	アイコン管理レコード	アイコン管理レコードそのもの
STRN	文字列情報	文字列 (終端コードはとくに必要ない)
PAT1	テキストタイプ 1 ページのビットイメージ	+\$00 バウンドレクタングル +\$08 テキストタイプ 1 ページのビットイメージ
PAT2	テキストタイプ 2 ページのビットイメージ	+\$00 バウンドレクタングル +\$08 テキストタイプ 2 ページのビットイメージ
PAT3	テキストタイプ 3 ページのビットイメージ	+\$00 バウンドレクタングル +\$08 テキストタイプ 3 ページのビットイメージ
PAT4	テキストタイプ 4 ページのビットイメージ	+\$00 バウンドレクタングル +\$08 テキストタイプ 4 ページのビットイメージ
mRGN	マスク用リージョンデータ	リージョンレコードそのもの
PAL2	パレットデータ	パレット 0~15 に対応するカラーコードを 1 ワードずつ。計 16 ワード。
TX16	パレットデータ付きテキストタイプ 4 ページのビットイメージ	+\$00 バウンドレクタングル +\$08 パレットデータ +\$28 テキストタイプ 4 ページのビットイメージ

7

そのほか

以上のような拡張/変更のほかに、ユーティリティ的機能を持つ SX コールが大量に追加されています。従来からの機能を利用するための便宜をはかるものが大半ですが、新しい概念を導入しているものもいくつか存在します。これらの概念について解説することにします。

●SX コールのベクタ

SX コールを呼び出す際、その番号に対応する処理ルーチンが呼び出されるわけですが、このとき、各ルーチンのアドレスが記録されているテーブルが参照されます。一般的に、このように参照されるルーチンのアドレスをベクタと呼びます。

SX1.02 までは、SX コールのベクタは参照することも変更することもできませんでした。ベクタが参照/変更できることで、ユーザは既存の SX コールをより付加価値の高い処理ルーチンに差し替えたり、バグを回避したりすることができるようになります。

SX1.10 では、ベクタの取得/変更を行うために、\$A422 SXGetVector、\$A423 SXSetVector が用意されています。

●FSX のロック

SX-SYSTEM を収めている FSX.X は、必要な場合にはメモリを広く使えるよう、常駐解除が可能となっています。これはこれで便利ではありますが、SX-WINDOW 動作中に常駐解除されてしまう可能性があり、その場合は確実に致命的なエラーが発生します。

SX1.10 では、FSX.X にロックレベルを導入し、ロックレベルが 1 以上の場合は FSX.X を常駐解除できないようにします。また、0 以下の場合は常駐解除を許可します。ロックレベルを 1 上げることを「ロックする」、1 下げたことを「アンロックする」と呼びます。

FSX のロック/アンロックを行う SX コールは、\$A42A SXLockFSX、\$A42B SXUnlockFSX です。

●画面モードへの対応

グラフィックマネージャが多様な画面モードに対応したのにもなって、すべてのマネージャの元締めでもあるタスクマネージャのレベル（≡シエルのレベル）でも画面モードの設定に対応するようになりました。\$A430 TSSetGraphMode を利用することによって、SX-WINDOW が動作する画面モードを設定することが可能です。

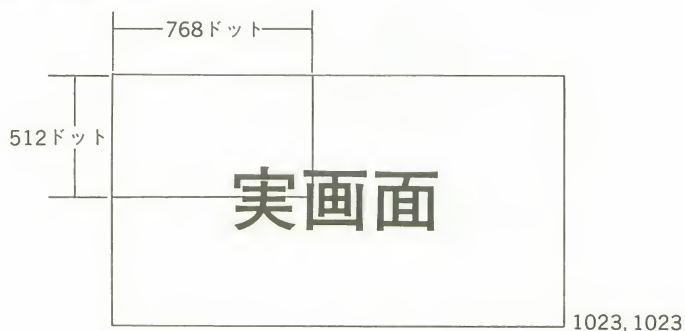
TSSetGraphMode では、IOCS の \$10 CRTMOD で指定するのと同様な数値によって画面モードを指定します。この値は SRAM に記録され、タスクマネージャが初期化される際に参照されてグラフィックマネージャに通知されます。これによって、以降、グラフィックマネージャは画面モードに適した描画を行うことができます。

さらに、TSSetGraphMode では、画面モードと同時に「実画面モード」にするかどうか

かの設定を行います。実画面モードを説明する前に、X68000 の画面の構造についておさらいしておきましょう。

X68000 の画面の大きさとグラフィック/テキスト VRAM の大きさは、かならずしも一致していません。たとえば、画面モード\$10, 「768×512 ドット, 16 色, グラフィック画面 1 ページ, 31kHz」のモードでは、実際にディスプレイ上で見ることが²できる領域は 768×512 ドットですが³, VRAM は 1024×1024 ドット分用意されています。前者を表示画面, 後者を実画面と呼びます。表示画面と実画面の関係を図にすると、図 7 のように表現できます。

■図 7 表示画面と実画面の関係



実画面モードは、表示画面の大きさに関係なく、実画面全体に描画を行うモードです。実画面モードでない場合は、表示画面の大きさの範囲内にのみ描画を行います。この機能はシェルにも反映されていて、15 ページのコラムに示したように、シェル起動時に画面モードを指定できるようになっています。

実画面内のどの部分を表示画面として表示するかは、CRTC のスクロールレジスタを操作することによって決めることができます。これを利用して、実画面モードに設定したうえで、マウスポインタの位置にあわせて表示画面の位置を操作するプログラムさえ用意すれば、実画面全体を広く利用することが可能となります。

COLUMN SRAM の内容

SX-WINDOW では、環境データの一部を SRAM に格納しています。

現時点では、SRAM のシステムが使用する領域 (\$ED0000~\$ED00FF) の一角, \$ED0072~\$ED007E を使用しています。この領域に収められている情報の形式は以下のとおりです。

\$ED0072.w	固定文字列 'SX'
\$ED0074.b	ダブルクリックの基準時間÷10
\$ED0075.b	マウススピード÷2
	以下はテキストパレットの設定。HSV方式をもとに格納されている
\$ED0076.b	色相 (パレット 0~3 共通)
\$ED0077.b	彩度 (パレット 0~3 共通)
\$ED0078.b	} パレット 0~3 の明度が5ビットずつ格納されている 形式は
\$ED0079.b	
\$ED007A.b	
	\$ED0078 \$ED0079 \$ED007A 00000111 11222223 3333xxxx
\$ED007B.b	使用するプリンタドライバ PRTD の ID (下位バイトのみ)
\$ED007C.b	bit7~bit4 SRAM 情報のバージョン
	bit1 画面状態保存モード
	bit0 スタート画面の保存モード
\$ED007D.b	デスクトップの背景を収めた PICT の ID (下位バイトのみ)
\$ED007E.b	画面モード
	bit7 1
	bit6 実画面モード
	bit5~bit0 画面モードを意味する数値

COLUMN | SYSDTOP.SX のフォーマット

SYSDTOP.SX は、シェルが管理するデスクトップ上の状態を保存しておくためのファイルです。システムアイコンの中の「スタート画面設定」を「する」に設定することで、シェルが終了した時点での状態が保存されます。

ファイルの形式は次のとおりです。

+ \$00.b	実画面モードフラグ (bit7 はつねに 1)
+ \$01.b	画面モード
+ \$02.w	タスク情報
+ \$00.w	タスクの数 (nt)
+ \$02	1 個目のタスク情報 (\$16e バイト)
+ \$00	タスク名 (ASCII2)
+ \$5a	コマンドライン (LASCII)
+ \$15a.1	ウィンドウコンテンツの左上ポイント (ローカル座標)
+ \$15e.1	ウィンドウコンテンツの右下ポイント (ローカル座標)
+ \$162.1	グローバル座標へのオフセットを意味するポイント
+ \$166.b	model
+ \$167.b	mode2
+ \$168.1	リソースタイプ
+ \$16c.w	リソース ID
:	
+ (nt * \$16e) + 4	アイコン情報

+\$00.l アイコンの個数 (ni)
+\$04 l 個目のアイコン情報
+\$00.b ユニット番号
+\$01.b メディアバイト
+\$02.l アイコンの左上のポイント
:

2⁴ その他のマネージャ

以上のマネージャ以外にも、小さな変更、追加が行われています。キーボードマネージャ、リソースマネージャ、ウィンドウマネージャ、メニューマネージャ、コントロールマネージャで、比較的目につく変更点について解説します。

1 キーボードマネージャ

キーボードマネージャのフラグの内容が一部変更になっています。

SX1.02 まで、フラグ Halt はキーデータを利用するか利用しないかを定めるためのフラグでしたが、意味が変更され、これが on (=1) になっている場合は、ショートカットキーとして [OPT.1] のほかに [XF2] が使えるようになります*7。

*7: キーダウンイベントの際のシフトキー情報で、[XF2] が押されているときに [OPT.1] に該当するビットを立てるようになります。

フラグ類を直接操作/変更する機能はいままで用意されていませんでしたが、\$A092 KBFlagGet、\$A093 KBFlagSet によって、それが可能となりました。これらの SX コールでは、1 ロングワードの各ビットにフラグの内容を割り当てた値が使われます。各ビットとフラグの関係は次のとおりです。

bit0	Halt
bit1	ResetOn
bit2	OldOn
bit3	LedOn
bit4	ClickOn
bit5	RepeatOn
bit6	AssignOn

2 リソースマネージャ

リソースマネージャには、次の 3 つの SX コールが追加されました。いずれもリソースの使用状況を調査するための SX コールです。

- \$A0ED RMResLinkGet

指定したリソースマップの次のリソースマップへのハンドルを得る。

- \$A0EE RMResTypeList

指定したリソースマップに登録されているリソースタイプの一覧を得る。

- \$A0EF RMResIDList

指定したリソースマップ中の、指定したタイプのリソースの ID の一覧を得る。

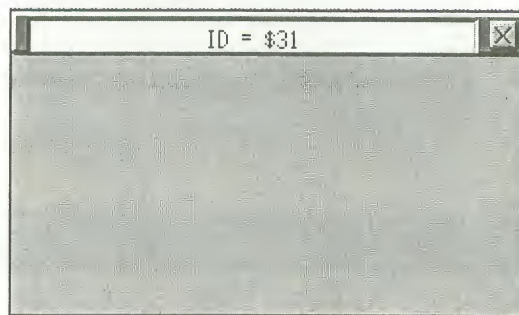
3 ウィンドウマネージャ

従来から用意されていた SX コールも、サブウィンドウとの関係から一部の機能が拡張されています。指定したウィンドウをアクティブにする \$A1FE WMSelect には、サブウィンドウリストからサブウィンドウを外す機能が追加されています (サブウィンドウマネージャの項を参照)。また、SX1.02 では未公開であった \$A1FF は、サブウィンドウリストを操作しないで同様な処理を行う WMSelect2 として公開されました。

SX コールの追加/変更以外では、新しいウィンドウの種類が 1 つ追加されています。ウィンドウ ID\$31 (49) の、このウィンドウ (図 8) は、標準ウィンドウとよく似ていますが、次の点で異なっています。

- ・ウィンドウアイテムとしては、クローズボタン、スクロールバー、サイズボタンをサポート
- ・タイトルバーのタイトルを表示する部分が広い

■図 8 ID\$31 のウィンドウ



むしろ、グラフィックをサポートする ID\$32 (50) のウィンドウから、グラフィックのサポートを取り除いたもの、と考えたほうがよいかもしれません。

このウィンドウは、新しい "コントロール.x" で使われています。

4 メニューマネージャ

メニューレコードの作成を補助するための SX コール、\$A269 MNConvert が追加されました。このコールでは、ある一定のフォーマットで書かれたメニュー定義用の ASCII 文字列を解析し、その内容にしたがってメニューレコードを作成してくれます。

メニュー定義文字列は、基本的にメニューアイテムを 1 つずつカンマで区切ったものですが、特殊文字を利用することによって、ショートカットやチェックマークの指定を行うことができます。

特殊文字には次の 3 つがあります。

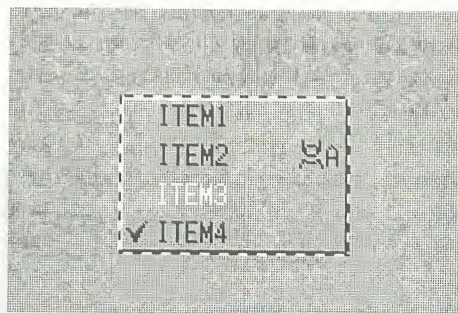
特殊文字	内 容
^	ショートカット文字の指定。次の 文字がショートカット文字となる
_	この文字で始まるアイテムはインアクティブとなる
!	チェックマークをつける

たとえば,

"ITEM1, ^AITEM2, ITEM3, !ITEM4"

という文字列であれば、図9のようなメニューを表示するメニューレコードが作成されます。

■図9 作成したメニューの表示例



5 コントロールマネージャ

コントロールマネージャには、コントロールオプション、ユーザ用のワークエリア、ドラッグ中に呼び出される関数のアドレス、そして定義関数のデータなど、コントロールレコードの中の値を参照/設定するための SX コールが追加されました。

2⁵ サンプルプログラム

本章で示したような SX1.10 の拡張/変更点を利用したサンプルプログラムをつくってみました。

1 プログラムの仕様

プログラム GRPSMPL は、データを与えることによって円グラフを作成します。
データはテキストファイルで、

<角度>, <欄名> 

の形式で 1 行ずつ各欄の要素を書いておきます。各要素は、円グラフ中の<角度>で示される扇型として色分けして表現されます。

角度は 0~360 の 10 進文字列、欄名は 16 バイト以内の文字列で表現します。全要素の角度の合計が 360 度になれば完全な円グラフになるわけですが、とくに値のチェックは行っていない。要素の数は (表現できる色の関係から *1) 15 個までです。

*1: もちろん、ペンパターンなどを駆使することによって、より多くの要素を表現できるわけですが、サンプルという性格上、そこまで行う必要はないと判断しました。

ファイルの名前をコマンドラインで指定するか、ファイルのアイコンをウィンドウ中にドラッグしてくることによってデータを与えてください。

ウィンドウ中のボタン [クリップボードへ] をクリックすると、ウィンドウ中に表示されているグラフのイメージとパレットがデスクトップスクラップに転送されます。これらの情報は Easypaint 等で利用することができます。

次のようなファイルを与えた場合、150 ページの図 1 のような表示が行われます。

```
30,X68000 (初代)
30,ACE
35,ACE-HD
30,PRO
32,PRO-HD
30,EXPERT
38,EXPERT-HD
18,PRO2
20,PRO2-HD
```

26,EXPERT2

28,EXPERT2-HD

10,SUPER

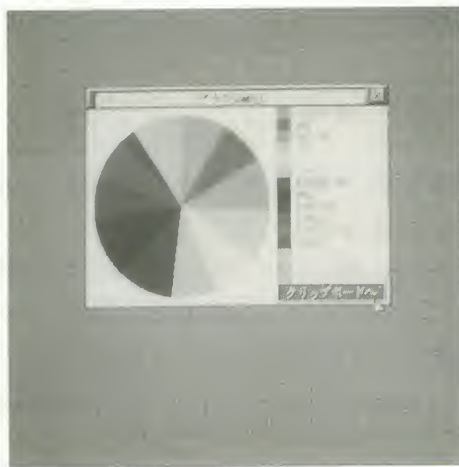
15,SUPER-HD

10,XVI

8,XVI-HD

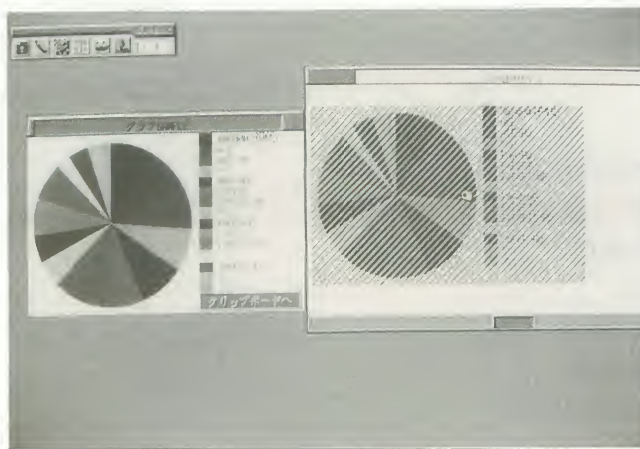
(このデータはサンプルであって根拠があるものではありません)

■図1 GRPSMPL.Xの表示例



Easypaintをお持ちの方は、デスクトップスクラップからイメージとパレットのペーストが可能であることを確認してください(図2)。

■図2 Easypaintとの関係



2

プログラムの説明

このプログラムで使用している SX1.10 の新しい機能には、次のようなものがあります。

・ID\$32 のウィンドウ

グラフィックをサポートする新しいウィンドウをオープンし、利用しています (GRPSMPL.S:258~269 行)。

この種のウィンドウでグラフィックタイプのビットマップを扱う場合、グラフポート先頭のビットマップレコードをテキストタイプ \longleftrightarrow グラフィックタイプに切り替えることで、ウィンドウ内部への描画をテキスト VRAM, グラフィック VRAM に振り分けることが可能です。その際の手順を参考になしてください (GRPSMPL.S:289~301, 326, 350~357, 368 行)*2。

*2:ビットマップの切り替えをよりエレガントに行いたい場合は、\$A1C6 GMExgBitmap を利用することをおすすめします。

・円弧 (Arc) の描画

いうまでもないことですが、円グラフの描画には円弧の塗り潰しを利用しています (GLPSMPL.S:522~526 行)。

・ビットの利用

描画した円グラフのイメージからセルに収めるデータを作成するために、ビットを利用して (GRPSMPL.S:600~670 行)。

セルの PAT4 形式はテキストタイプのレクタングルイメージですが、GRPSMPL ではグラフィックタイプのビットマップへの描画を行っています。この描画で得られるイメージはグラフィックタイプですし、そのイメージにしても、直接 VRAM を参照して得るのはスマートな方法とはいえません。そこで、描画の手順をスクリプトに記録しておき、それを画面に描画する場合はグラフィックタイプのビットマップに、セルのデータを作成する場合はテキストタイプのビットに描画しています。これにより、ビット内にはセルのデータの元となるテキストタイプのイメージが作成されることになるわけです*3。

*3:\$A1BB GMTTransImg を利用することによって、異なるタイプのビットイメージ/ビット間でイメージのやりとりをすることができますが、ここでは再表示の容易さ等の関係からスクリプトを利用しています。

ビット中にスクリプトを描画するために、ビットと結び付けられたグラフポートを作成している部分も参考にしてください (GRPSMPL.S:593~598, 611~619, 672~675 行)。

・セルの新タイプ PAT4 と PAL2

デスクトップスクラップに画像データとパレットデータを収めるために、新タイプ PAT4 と PAL2 を利用しています (GRPSMPL.S:626~666 行)。

セルリスト格納用の再配置可能ブロックを作成し (GRPSMPL.S:633~636 行), その中にタイプ名 PAT4, PAT4 のサイズ, バウンドレクタングル, そしてピッツ内のテキストタイプイメージを格納します (GRPSMPL.S:640~649 行)。続いて, タイプ名 PAL2, PAL2 のサイズ, パレットデータを格納し (GRPSMPL.S:651~657 行), このブロックをデスクトップスクラップに収めています (GRPSMPL.S:659~662 行)。スクラップへ転送した後は, セルリストを収めたブロックは不要になるので, 廃棄します (GRPSMPL.S:664~666 行)。

3

プログラムリスト

このプログラムの場合, コマンドラインからファイル名を受け取るために, スケルトンの一部に手を加えてあります (SKELTON.S:78 行)。実行ファイル作成の際は注意してください。

リスト 1 に SKELTON.S, リスト 2 に WORK.INC, リスト 3 に GRPSMPL.S を示します。また, リスト 4 に実行ファイルを作成するための makefile を示します。

■リスト 1 GRPSMPL 用 SKELTON.S

```

1 *
2 *          SX-WINDOW
3 *          GRPSMPL
4 *
5 *          スケルトン
6 *
7
8          .include      DOSCALL.MAC
9          .include      SXCALL.MAC
10
11          .xref      _INIT,_TINI
12          .xref      TOLE
13          .xref      MSLDOWN,MSLUP,MSRDOWN,MSRUP
14          .xref      KEYDOWN,KEYUP
15          .xref      UPDATE,ACTIVATE
16          .xref      SYSTEM1,SYSTEM2,SYSTEM3,SYSTEM4
17
18          .include      WORK.INC          * ワークエリアの内容を
                                           * 定義するファイル
19
20          .text
21 mdhead:
22          dc.l      '0BJR'
23          dc.l      0
                                           * [ モジュールヘッダ ]
                                           * R型モジュール
                                           * プログラムエリアのサイズ
                                           * (Xファイルの場合
                                           * 意味がない)
                                           * スタートアドレスオフセット
                                           * ワークエリアのサイズ
                                           * システム予約
24          dc.l      _main-mdhead
25          dc.l      WORKSIZE+STKSIZE
26          dc.l      0,0,0,0
27
28 DiXstart:                                * コマンドラインから

```

```

29
30
31      lea     64(a1), a1
32      move.l  a1, sp
33      lea     16(a0), a0
34      sub.l   a0, a1
35      move.l  a1, -(sp)
36      pea     (a0)
37      DOS     _SETBLOCK
38
39      clr.l   -(sp)
40      pea     comm(pc)
41      pea     shname(pc)
42      move.w  #2, -(sp)
43      DOS     _EXEC
44
45      clr.l   -(sp)
46      pea     comm(pc)
47      pea     shname(pc)
48      clr.w  -(sp)
49      DOS     _EXEC
50
51      tst.l   d0
52      bpl     p_execil
53
54      pea     mes_execerr(pc)
55      DOS     _PRINT
56
57      .data
58      mes_execerr:
59      dc.b    'カーネルの起動に失敗しました!!!', 13, 10, 0
60      .even
61      shname:
62      dc.b    'SXKERNEL.X -K -R7 -L1', 0
63      ds.b    70
64      .even
65      .bss
66      comm:
67      ds.b    258
68
69
70
71      .text
72      _main:
73      movea.l a1, a5
74      move.l  a2, cmdLine(a5)
75      move.l  a3, envPtr(a5)
76
77      clr.w   -(sp)
78      pea.l   name(a5)
79      pea.l   winRect(a5)
80      pea.l   (a2)
81      SXCALL  $A3EA
82      lea.l   14(sp), sp
83      move.w  d0, paramFlg(a5)
84
85      bsr     _INIT
86      bmi     _exit
87

```

* 起動した場合
 * ここからスタートする

* 専有メモリを縮小する

* デバッグ用カーネルの
 * バスをサーチする

* デバッグ用カーネルを
 * 立ち上げる

* 正常に終了した場合
 * そのまま終了

* エラーメッセージを
 * 表示する

* 終了

* カーネルはここから先の
 * コードを読み込み、
 * タスクとして立ち上げる

* SX-SHELLから起動した場合
 * ここからスタートする

* ## ここが標準と違う ##

* _TSTakeParam
 * コマンドラインを解析し、'-W'
 * オプションを得る

* アプリケーションの初期化
 * 初期化時にエラーがあれば
 * 終了

```

88      move.w    #$ffff, eventMask(a5)
89 loop:
90      pea      eventRec(a5)
91      move.w    eventMask(a5), -(sp)
92      SXCALL    $A357
93      addq.l    #6, sp
94      lea      eventTable(pc), a1
95      move.w    eventRec_what(a5), d0
96      and.w     #15, d0
97      add.w     d0, d0
98      move.w    (a1, d0.w), d0
99      jsr      (a1, d0.w)
100     tst.l     d0
101     bmi      _exit
102     bra      loop
103
104 eventTable:
105     dc.w      IDLE-eventTable
106     dc.w      MSLDOWN-eventTable
107     dc.w      MSLUP-eventTable
108     dc.w      MSRDOWN-eventTable
109     dc.w      MSRUP-eventTable
110     dc.w      KEYDOWN-eventTable
111     dc.w      KEYUP-eventTable
112     dc.w      UPDATE-eventTable
113     dc.w      DAMMY-eventTable
114     dc.w      ACTIVATE-eventTable
115     dc.w      DAMMY-eventTable
116     dc.w      DAMMY-eventTable
117     dc.w      SYSTEM1-eventTable
118     dc.w      SYSTEM2-eventTable
119     dc.w      SYSTEM3-eventTable
120     dc.w      SYSTEM4-eventTable
121
122 DAMMY:
123     rts
124
125 _exit:
126     bsr      _TINI
127
128     move.w    d0, -(sp)
129     SXCALL    $A352
130
131     .end      DiXstart
132

```

メインループ

TSEventAvail
イベントを得る

イベントコードによって
分岐する

分岐先のテーブル

0 アイドルイベント
1 レフトダウンイベント
2 レフトアップイベント
3 ライトダウンイベント
4 ライトアップイベント
4 キーダウンイベント
6 キーアップイベント
7 アップデートイベント
8 ー
9 アクティブイベント
10 ー
11 ー
12 システムイベント 1
13 システムイベント 2
14 システムイベント 3
15 システムイベント 4

[終了する]
アプリケーションの
終了処理

TSExit

■リスト 2 GRPSMPL 用 WORK.INC

```

1 #
2 #
3 #
4 #
5 #
6 #
7
8 STKSIZE      =      2*1024
9
10 #
11
12 .offset 0
13 cmdLine:

```

SX-WINDOW
GRPSMPL

ワーク定義用インクルードファイル

スタックサイズ

ワークの内容の定義

コマンドラインの

14		ds. l	1	* アドレスを
15	envPtr:			* 保存するワーク
				* 環境のアドレスを
				* 保存するワーク
16		ds. l	1	
17	name:			* コマンドラインで指定された
18		ds. b	90	* ファイル名が入るバッファ
19	winRect:			* ウィンドウレクタングル
				* レコード
20		ds. l	2	
21	paramFlg:			* コマンドラインの
				* 解析結果を示す
22		ds. w	1	* フラグ
23	eventRec:			* イベントレコードの先頭
24	eventRec_what:			* イベントコード
25		ds. w	1	
26	eventRec_whom1:			* 第1引数
27		ds. l	1	
28	eventRec_when:			* イベント発生時
29		ds. l	1	
30	eventRec_whom2:			* 第2引数
31		ds. l	1	
32	eventRec_what2:			
33		ds. w	1	* タスクマネージャ
				* イベントの種類
34	eventRec_taskID:			
35		ds. w	1	* 送り手のタスクID
36	eventMask:			* イベントマスクを
				* 保存するワーク
37		ds. w	1	
38	taskID:			* タスクIDを保存するワーク
39		ds. l	1	
40	winPtr:			* ウィンドウレコードを
				* 作成する場所
41		ds. b	\$72	
42				
43	winActive:			* アクティブフラグ
44		ds. w	1	
45				
46	bitmapRec:			* グラフィックタイプの
				* ビットマップ
47		ds. b	\$16	* レコードを収める場所
48	ctrlHdl:			
49		ds. l	1	* コントロールレコード
				* へのハンドル
50	rgnHdl:			
51		ds. l	1	* 汎用リージョンへの
				* ハンドル
52	scriptHdl:			
53		ds. l	1	* スクリプトへのハンドル
54	bitsHdl:			
55		ds. l	1	* ビッツへのハンドル
56				
57	angleList:			
58		ds. w	15	* 角度のリスト
59	tagList:			
60		ds. w	16*15	* 欄名のリスト
61				
62				
63	WORKSIZE:			* ワークの終了
64				

■リスト3 GRPSMPL.S

```

1  *
2  *          SX-WINDOW
3  *          GRPSMPL
4  *
5  *          初期化&終了&イベント処理モジュール
6  *
7
8          .include      DOSCALL.MAC
9          .include      IOCALL.MAC
10         .include      SXCALL.MAC
11
12         .xdef      _INIT, _TINI
13         .xdef      IDLE
14         .xdef      MSLDOWN, MSLUP, MSRDOWN, MSRRUP
15         .xdef      KEYDOWN, KEYUP
16         .xdef      UPDATE, ACTIVATE
17         .xdef      SYSTEM1, SYSTEM2, SYSTEM3, SYSTEM4
18
19         .include      WORK.INC          * ワークエリアの内容を
                                           * 定義するファイル
20
21 RGB          macro      R, G, B, I          * カラーコード変換用マクロ
22          dc.w          (G.shl.11) + (R.shl.6) + (B.shl.1) + I
23          endm
24
25 WINOPT      =          %0000          * ウィンドウオプション
26 WIN_X      =          320          * ウィンドウ初期x
27 WIN_Y      =          200          * ウィンドウ初期y
28
29         .text
30 MSLUP:
31 MSRDOWN:
32 MSRRUP:
33 KEYDOWN:
34 KEYUP:
35 SYSTEM3:
36 SYSTEM4:
37          moveq      #0, d0          * [ レフトアップイベント ]
                                           * [ ライトダウンイベント ]
                                           * [ ライトアップイベント ]
                                           * [ キーダウンイベント ]
                                           * [ キーアップイベント ]
                                           * [ システムイベント3 ]
                                           * [ システムイベント4 ]
                                           * 以上のイベントでは
                                           * なにもしない
38          rts
39
40 IDLE:
41          moveq      #0, d0          * [ アイドルイベント ]
42          rts
43
44 MSLDOWN:
45          move.l      eventRec_whoml(a5), a0          * [ レフトダウンイベント ]
46          lea         winPtr(a5), a2          * 自分のウィンドウ上で
47          cmp.l       a2, a0          * 発生したか?
48          bne         MSLD9          * 違うならMSLD9へ
49
50          tst.b       winActive(a5)          * 現在ウィンドウは
                                           * アクティブか?
                                           * アクティブならMSLD1へ
51          bne         MSLD1
52          pea         (a2)
53          SXCALL      $A1FE          * __WMSelect
54          addq.l      #4, sp
55          bra         MSLD9          * アクティブにするだけ
56 MSLD1:
57          pea         eventRec(a5)
58          pea         (a2)
59          SXCALL      $A3A2          * ウィンドウ処理
                                           * __SXCallWindM

```

```

60      addq.l  #8, sp
61      tst.l   d0
62      beq     MSLD9
63
64      cmp.w   #7, d0
65      beq     CloseBttn
66
67      clr.l   -(sp)
68      clr.l   -(sp)
69      clr.l   -(sp)
70      pea     eventRec(a5)
71      pea     (a2)
72      SXCALL  $A3A3
73      lea     20(sp), sp
74
75      cmp.w   #10, d0
76      bne     MSLD9
77
78      bsr     Copy2Scrap
79 MSLD9:
80      moveq    #0, d0
81      rts
82
83 CloseBttn:
84      moveq    #-1, d0
85      rts
86
87
88 UPDATE:
89      pea     winPtr(a5)
90      SXCALL  $A20D
91      addq.l  #4, sp
92
93      bsr     DrawGraph
94
95      pea     winPtr(a5)
96      SXCALL  $A20E
97      addq.l  #4, sp
98
99      moveq    #0, d0
100     rts
101
102 ACTIVATE:
103     move.l   eventRec_whoml(a5), d0
104     beq      ACT9
105     lea      winPtr(a5), a0
106     cmp.l    a0, d0
107     bne      ACT0
108     tst.b    winActive(a5)
109     bne      ACT9
110
111     st       winActive(a5)
112     bsr      SetPalet
113     bra      ACT9
114 ACT0:
115     sf       winActive(a5)
116 ACT9:
117     moveq    #0, d0
118     rts
119
120 SYSTEM1:
121 SYSTEM2:
122     move.w   eventRec_what2(a5), d0
123     cmp.w    #1, d0

```

```

* どこも操作されなかった?
*   ならばMSLD9へ

* クローズボタン?
*   ならばCloseBttnへ

* コントロール処理
*   _SXCallCtrlM

* 標準ボタンが押された?
*   違うのならMSLD9へ

* [ アップデートイベント ]
*   _WMUpdate
*   アップデート開始

* ウィンドウ内部を描画
*   _WMUpdtOver
*   アップデート終了

* [ アクティブイベント ]
*   自分のウィンドウが
*   アクティブになった?
*   違うのならACT0へ

* アクティブフラグをセット
*   パレットを再設定

* アクティブフラグをリセット

* [ システムイベント1 ]
* [ システムイベント2 ]

* タスクの終了?

```

```

124      beq      AllClose
125      cmp.w   #2, d0
126      beq      AllClose
127      cmp.w   #31, d0
128      beq      Save
129      cmp.w   #32, d0
130      beq      WindowSelect
131      cmp.w   #81, d0
132      beq      DragEnd
133
134      moveq    #0, d0
135      rts
136
137 AllClose:
138      moveq    #-1, d0
139      rts
140
141 WindowSelect:
142      pea      winPtr (a5)
143
144      SXCALL   $A1FE
145      addq.l   #4, sp
146
147      moveq    #0, d0
148      rts
149
150 Save:
151      link     a6, #-512
152
153      move.w   #-1, -(sp)
154      pea      -512(a6)
155      SXCALL   $A35B
156      addq.l   #6, sp
157
158      lea      -512+$5a(a6), a1
159
160      move.l   a1, a2
161      addq.l   #1, a2
162      lea      name(a5), a0
163      moveq    #-1, d1
164      moveq    #90-1, d0
165
166 Save0:
167      addq.l   #1, d1
168      move.b   (a0)+, (a2)+
169      dbeq     d0, Save0
170
171      move.b   d1, (a1)
172
173      move.w   #-1, -(sp)
174      pea      -512(a6)
175      SXCALL   $A35C
176      addq.l   #6, sp
177
178      unlink   a6
179      moveq    #0, d0
180      rts
181
182 DragEnd:
183      lea      winPtr(a5), a2
184      move.l   eventRec_whom1(a5), a0
185      cmp.l    a2, a0
186
187      bne      DragEnd9

```

```

*   ならばLetsGoAwayへ
*   全ウィンドウのクローズ?
*   ならばLetsGoAwayへ
*   セーブ?
*   ならばSaveへ
*   ウィンドウのセレクト?
*   ならばWindowSelectへ
*   ドラッグ終了?
*   ならばDragEndへ

*   現在の状態をセーブ

*   自分のタスク管理テーブルを
*   コピーしてくる
*   __TSGetTdb

*   A1 : コマンドライン
*   (LASCII)の先頭

*   A2 : コマンドライン文字列
*   表示中のファイル名

*   コマンドラインにコピー

*   文字数を収める

*   自分のタスク管理テーブルに
*   コピーする
*   __TSSetTdb

*   [ドラッグ終了イベント処理]

*   自分のウィンドウ上
*   で離された?
*   違うならDragEnd9へ

```



```

185      pea      (a2)                                * 自分のウィンドウを
186      SXCALL   $A131                                * カレントに
187      addq. l  #4, sp                                * __GMSetGraph
188
189      SXCALL   $A38D                                * __TSHideDrag
190
191      SXCALL   $A389                                * __TSGetDrag
192      bne      DragEnd8                              * データがなければ
193                                           * DragEnd8へ
194
195      move. l  (a0), d1                                * セルリスト長
196      move. l  4(a0), a1                              * セルリストへのハンドル
197      move. l  (a1), a0                              * セルリストのアドレス
198 DragEnd0:
199      cmp. w   #'FS', (a0)                            * アイコン管理レコード?
200      beq      DragEnd1                              *   ならばDragEnd1へ
201
202      move. l  4(a0), d0                                * 次のセルへ
203      addq. l  #8, d0                                *   これで終わり?
204      lea      (a0, d0, 1), a0                        *   でなければDragEnd0へ
205      sub. l   d0, d1                                *   終わりならDragEnd8へ
206      bhi      DragEnd0
207      bra      DragEnd8
208 DragEnd1:
209      addq. l  #8, a0
210
211      pea      name(a5)                                * アイコン管理レコードから
212      pea      (a0)                                    * フルバスのファイル名を得る
213      SXCALL   $A3BB                                * __TISISRecToStr
214      addq. l  #8, sp
215
216      bsr      ReadAndDraw                            * ドラッグされたファイルを
217                                           * 読み込み、
218                                           * 表示を行なう
219
220      move. w   d0, -(sp)
221      SXCALL   $A38C                                * __TSEndDrag
222      addq. l  #2, sp
223      bra      DragEnd9
224 DragEnd8:
225      move. w   #1, -(sp)                            * はじき返す
226      SXCALL   $A38C                                * __TSEndDrag
227      addq. l  #2, sp
228 DragEnd9:
229      moveq     #0, d0
230      rts
231 _INIT:
232                                           * [ アプリケーションの
233                                           * 初期化を行なう ]
234
235      move. l  winRect(a5), d0
236      move. w   paramFlg(a5), d1
237      btst     #0, d1
238
239      beq      _INIT0                                * '-W' オプションが
240                                           * 指定された?
241                                           *   指定されていないければ
242                                           *   _INIT0へ
243
244      move. l  winRect+4(a5), d1
245      beq      _INIT1
246      tst. w   d1
247      cmp. w   d0, d1
248      ble      _INIT1
249      swap     d0
250      swap     d1

```

```

243      cmp.w    d0, d1
244      bgt      _INIT2
245      swap     d0
246      swap     d1
247      bra      _INIT1
248 _INIT0:
249      SXCALL    $A35E      * _TSGetWindowPos
250      move.l    d0, winRect(a5) * デフォルト位置を得る
251 _INIT1:
252      add.l      #WIN_X+$10000+WIN_Y, d0 * ウィンドウレクタングルを
* 作成
253      move.l     d0, winRect+4(a5)
254 _INIT2:
255      SXCALL    $A360      * _TSGetID
256      move.l    d0, taskId(a5) * タスクIDを得る
257
258      move.l     d0, -(sp)    * タスクID
259      move.w     #-1, -(sp)   * クローズボックスあり
260      move.l     #-1, -(sp)   * もっとも手前に
261      move.w     #$32*16+WINOPT, -(sp) * ウィンドウID=$32
262      move.w     #-1, -(sp)   * 可視
263      pea.l      winTitle(pc) * ウィンドウタイトル
264      pea.l      winRect(a5)  * ウィンドウレクタングル
265      pea        winPtr(a5)   * ワーク上に作成
266      SXCALL    $A1F9      * _WMOpen
267      lea.l      26(sp), sp   * ウィンドウを開く
268      tst.l      d0          * エラー?
269      bmi        _INIT_Err   * ならば_INIT_Errへ
270
271      st         winActive(a5) * アクティブフラグをセット
272
273      bsr        DrawGraph1st * ウィンドウ内部を描画する
274      * (最初の1回)
275      move.w     paramFlg(a5), d0
276      btst      #1, d0
* ファイル名が
* 指定されている?
* いなければ_INIT9へ
* 指定されたファイルを
* 読み込み、
* 表示する。
277      beq        _INIT9
278
279      bsr        ReadAndDraw
280 _INIT9:
281      moveq      #0, d0
282      rts
283 _INIT_Err:
284      moveq      #-1, d0
285      rts
286
287 DrawGraph1st:
288      * ウィンドウ内部の描画の
289      * 準備をするサブルーチン
290      pea        bitmapRec(a5)
291      move.w     #1, -(sp)
292      SXCALL    $A16D      * _GMinInitBitmap
293      addq.l     #6, sp
294
295      lea        winPtr(a5), a1
296      lea        bitmapRec(a5), a2
297
298      move.l     (a1), a0
299      * カレントグラフィポートの
300      * ビットマップを
301      * グラフィック画面用のものに
302      * 差し替え、グラフィック画面
303      * への描画を始める

```

```

303      pea      (a1)                                * 自分のウィンドウ
304      SXCALL   $A131                                * (グラフィック画面)
305      addq. l   #4, sp                                * __GMSetGraph
306
307      SXCALL   $A15A                                * __GMNewRgn
308      move. l   a0, rgnHdl (a5)                      * 汎用リージョン
309
310      pea      winLocalRect (pc)                      * ウィンドウ内部を意味する
311      pea      (a0)                                * レクタングルリージョンを
312                                          * 作成
313      SXCALL   $A15F                                * __GMRectRgn
314      addq. l   #8, sp
315
316      pea      (a0)                                * 初期スクリプト作成開始
317      SXCALL   $A199                                * __GMOpenScript
318      addq. l   #4, sp
319
320      bsr      CLS                                    * 画面を塗り潰す
321                                          * (スクリプトに記録)
322
323      clr. l    -(sp)                                * 記録終了
324      SXCALL   $A19A                                * __GMCloseScript
325      addq. l   #4, sp
326      move. l   a0, scriptHdl (a5)
327
328      move. l   a3, (a1)                                * カレントビットマップを
329      pea      (a1)                                * テキスト画面へ
330                                          * 自分のウィンドウ
331      SXCALL   $A131                                * (テキスト画面)
332      addq. l   #4, sp                                * __GMSetGraph
333
334      move. l   #0, -(sp)                                * 標準ボタンを作成する
335      move. w   #0*16, -(sp)                          * 標準ボタン
336      move. w   #1, -(sp)                                * 最大値は1
337      move. w   #0, -(sp)                                * 最小値は0
338      move. w   #0, -(sp)                                * 初期値は0
339      move. w   #-1, -(sp)                              * 可視
340      pea      cTitle (pc)                            * タイトル
341      pea      cRect (pc)                             * 配置する位置と大きさ
342      pea      (a0)                                * 自分のウィンドウ上に配置
343      SXCALL   $A289                                * __GMOpen
344      lea      26 (sp), sp
345      move. l   a0, ctrlHdl (a5)
346
347      bsr      SetPalet                                * グラフィックパレットを
348                                          * 設定
349
350      rts
351
352      DrawGraph:
353      lea      winPtr (a5), a1
354      lea      bitmapRec (a5), a2
355
356      move. l   (a1), a0                                * カレントグラフポートの
357                                          * ビットマップを
358      move. l   2 (a0), bitmapRec+2 (a5)              * グラフィック画面用のものに
359      move. l   6 (a0), bitmapRec+6 (a5)              * 差し替え、グラフィック画面
360      move. l   a0, a3                                * への描画を始める
361      move. l   a2, (a1)
362
363      pea      (a1)                                * 自分のウィンドウ

```

360	SXCALL	\$A131	* (グラフィック画面)
361	addq. l	#4, sp	* __GMSetGraph
362			
363	pea	winLocalRect (pc)	* 記録しておいたスクリプト
364	move. l	scriptHdl (a5), -(sp)	* を描画する
365	SXCALL	\$A19C	* __GMDrawScript
366	addq. l	#4, sp	
367			
368	move. l	a3, (a1)	* カレントビットマップを
			* テキスト画面へ
369	pea	(a1)	* 自分のウィンドウ
			* (テキスト画面)
370	SXCALL	\$A131	* __GMSetGraph
371	addq. l	#4, sp	
372	pea	(a1)	* ウィンドウ上の
			* コントロールを描画
373	SXCALL	\$A28E	* __CMDraw
374	addq. l	#4, sp	
375			
376	rts		
377			
378	CLS:		* [画面を塗り潰す
			* サブルーチン]
379	move. w	#15, -(sp)	* バックグラウンドカラー
			* = 15
380	SXCALL	\$A148	* __GMBackColor
381	addq. l	#2, sp	
382	move. w	#\$100, -(sp)	* バックグラウンドカラー
			* PSET
383	SXCALL	\$A144	* __GMPenMode
384	addq. l	#2, sp	
385	pea	winLocalRect (pc)	* ウィンドウ内部を塗り潰す
386	SXCALL	\$A173	* __GMFillRect
387	addq. l	#4, sp	
388	move. w	#\$00, -(sp)	* PSET
389	SXCALL	\$A144	* __GMPenMode
390	addq. l	#2, sp	
391	move. w	#\$00, -(sp)	* PSET
392	SXCALL	\$A18D	* __GMFontMode
393	addq. l	#2, sp	
394			
395	rts		
396			
397	_TINI:		* [終了処理]
398	move. l	rgnHdl (a5), -(sp)	* 汎用リージョンを廃棄
399	SXCALL	\$A15B	* __GMDisposeRgn
400	addq. l	#4, sp	
401			
402	move. l	scriptHdl (a5), -(sp)	* スクリプトを廃棄
403	SXCALL	\$A19B	* __GMDisposeScript
404	addq. l	#4, sp	
405			
406	pea	winPtr (a5)	* ウィンドウ上の
			* コントロールを廃棄
407	SXCALL	\$A28B	* __CMKill
408	addq. l	#4, sp	
409			
410	pea	winPtr (a5)	* ウィンドウをクローズする
411	SXCALL	\$A1FB	* __WMClose
412	addq. l	#4, sp	* WMDisposeでないことに注意
413			
414	moveq	#0, d0	
415	rts		


```

416
417
418 SetPalet:                                     * [ グラフィック
                                                * パレットを設定する ]

419         lea     paletData(pc), a1
420         moveq    #0, d1
421         moveq    #16-1, d3
422 SetPalet0:
423         move.w   (a1)+, d2
424         lOCS     _GPALET
425         addq.l   #1, d1
426         dbra     d3, SetPalet0
427
428         rts
429
430 ReadAndDraw:                                   * [ ファイル読み込み
                                                * &スクリプト作成 ]
                                                * READ

431         move.w   #0, -(sp)
432         pea      name(a5)
433         DOS       _OPEN
                                                * オープン
434         addq.l   #6, sp
435         move.l   d0, d7
                                                * エラー?
436         bmi      RD_fileErr
                                                *   ならばRD_fileErrへ
437
438         move.l   #' ', d5
                                                * 定数(スペース4つ)
439         lea      angleList(a5), a3
440         lea      tagList(a5), a4
441         moveq    #15-1, d6
                                                * 要素は15個まで
442 ReadAndDraw0:
443         move.w   d7, -(sp)
444         pea      inpPtr(pc)
445         DOS       _FGETS
                                                * 1行読み込み
446         addq.l   #6, sp
447         tst.l    d0
                                                * もう読めない?
448         bmi      ReadAndDraw3
                                                *   ならばReadAndDraw3へ
449
450         lea      inpPtr+2(pc), a0
451         moveq    #0, d0
452         move.b   -1(a0), d0
453         sf       (a0, d0)
                                                * ASCII文字列に変換
                                                * (念のため)
454
455 ReadAndDraw1:
456         move.l   a0, a1
457         move.b   (a1)+, d0
                                                * カンマを探して$00に
                                                * 置き換える
458         beq      RD_formErr
                                                *   文字列終端ならば
                                                *   RD_formErrへ
459         cmp.b    #' ', d0
                                                * カンマ?
460         bne      ReadAndDraw1
                                                *   でなければ
                                                *   ReadAndDraw1へ
461
462         sf       -1(a1)
                                                * $00に置き換えて、
                                                * 10進文字列の部分を
463         dc.w     $FE10
                                                * FPACK __STOL
464         move.w   d0, (a3)+
                                                * で数値に変換、
                                                * バッファに格納する
465
466         move.l   d5, (a4)
                                                * 欄名バッファを初期化
467         move.l   d5, 4(a4)
468         move.l   d5, 8(a4)
469         move.l   d5, 12(a4)
470         move.l   a4, a0
471         moveq    #16-1, d1
                                                * 16文字まで

```

```

472 ReadAndDraw2:
473     move. b    (a1) +, (a4) +
474     dbeq      d1, ReadAndDraw2
475     move. b    #$20, -1 (a4)
476     lea       16 (a0), a4
477
478     dbra      d6, ReadAndDraw0      * 15個読み終わるまでループ
479     bra       ReadAndDraw4         * 読み終わったら
                                        * ReadAndDraw4へ

480
481 ReadAndDraw3:
482     clr. w     (a3) +                * 残りの要素をクリア
483     move. l    d5, (a4) +
484     move. l    d5, (a4) +
485     move. l    d5, (a4) +
486     move. l    d5, (a4) +
487     dbra      d6, ReadAndDraw3
488
489 ReadAndDraw4:
490     move. w    d7, -(sp)
491     DOS        _CLOSE
492     addq. l    #2, sp                * クローズ
493
494     move. l    scriptHdl (a5), -(sp) * これまでのスクリプトを廃棄
495     SXCALL     $A19B
496     addq. l    #4, sp                * __GMDisposeScript
497
498     move. l    rgnHdl (a5), -(sp)    * 新たにスクリプトを記録開始
499     SXCALL     $A199
500     addq. l    #4, sp                * __GMOpenScript
501
502     bsr       CLS                    * まず画面の初期化を記録
503
504     move. w    #90, d5                * 90° からスタート
505     lea       angleList (a5), a3
506     lea       tagList (a5), a4
507     moveq     #15-1, d6
508 ReadAndDraw5:
509     move. l    d5, d4
510     move. w    (a3) +, d0
511     beq       ReadAndDraw7
512     sub. w    d0, d4
513     bpl       ReadAndDraw6
514     add. w    #360, d4
515 ReadAndDraw6:
516     moveq     #14, d1
517     sub. w    d6, d1
518     move. w    d1, -(sp)              * 描画色を決める
519     SXCALL     $A147                  * __GMForeColor
520     addq. l    #2, sp
521
522     move. w    d5, -(sp)
523     move. w    d4, -(sp)              * 数値にしたがって各要素の
524     pea       ovalRect (pc)           * 円弧を塗り潰して描画する
525     SXCALL     $A179                  * __GMFillArc
526     addq. l    #8, sp
527
528     addq      #1, d1
529     mulu      #12, d1
530     swap     d1
531     move. w    #216, d1
532     swap     d1
533     move. l    d1, -(sp)
534     sub. l    #$0010_000c, d1

```

```

535      move.l  d1, -(sp)
536      pea     (a7)
537      SXCALL  $A173
538      lea     12(sp), sp
539
540      add.l   #$0016_0000, d1
541      move.l  d1, -(sp)
542      SXCALL  $A16E
543      addq.l  #4, sp
544      move.w  #16, -(sp)
545      clr.l   -(sp)
546      pea     (a4)
547      SXCALL  $A191
548      lea     10(sp), sp
549  ReadAndDraw7:
550      lea     16(a4), a4
551      move.l  d4, d5
552      dbra    d6, ReadAndDraw5
553
554      clr.l   -(sp)
555      SXCALL  $A19A
556      addq.l  #4, sp
557      move.l  a0, scriptHdl(a5)
558
559      bsr     DrawGraph
560
561      moveq   #0, d0
562      rts
563
564  RD_fileErr:
565      pea     fileErrMsg(pc)
566      move.w  #1, -(sp)
567      SXCALL  $A2F6
568      addq.l  #6, sp
569
570      moveq   #1, d0
571      rts
572
573  RD_formErr:
574      pea     formErrMsg(pc)
575      move.w  #1, -(sp)
576      SXCALL  $A2F6
577      addq.l  #6, sp
578
579      move.w  d7, -(sp)
580      DOS     _CLOSE
581      addq.l  #2, sp
582
583      moveq   #1, d0
584      rts
585
586  Copy2Scrap:
587      link    a6, #-40
588
589      clr.l   -(sp)
590      SXCALL  $A0B7
591      addq.l  #4, sp
592
593      pea     -40(a6)
594      move.w  #0, -(sp)
595
596      SXCALL  $A12D
597      addq.l  #6, sp

```

```

# 右側の色見本
# __GMFillRect

```

```

# __GMMove

```

```

# 欄名を描画
# __GMDrawStr

```

```

# 15個描くまで繰り返す

```

```

# スクリプト記録終了
# __GMCloseScript

```

```

# 記録したスクリプト
# を描画する

```

```

# ドラッグを受け入れる

```

```

# ファイルが読めない旨
# エラーダイアログで警告
# __DMError

```

```

# ドラッグははじき返す

```

```

# ファイルの形式が違う旨
# エラーダイアログで警告
# __DMError

```

```

# クローズ

```

```

# ドラッグははじき返す

```

```

# [ データをスクラップへ ]

```

```

# 踏切ポインタへ
# __EMEnCross

```

```

# テキストタイプの
# グラフポート
# __GMOpenGraph

```

```

597      move.l    a0, a2
598      move.l    (a2), a3
599
600      move.w    #%1111, -(sp)      * 4ページ
601      pea      winLocalRect(pc)    * 大きさはウィンドウと同じ
602      clr.w     -(sp)              * テキストタイプのビット
603      SXCALL    $A1CA              * __GMNewBits
604      addq.l    #8, sp
605      move.l    a0, bitsHdl(a5)
606      pea      (a0)                * ロックして使用開始
607      SXCALL    $A1CC              * __GMLockBits
608      addq.l    #4, sp
609      move.l    (a0), a1
610
611      move.l    a1, (a2)            * 作成したグラフポートと
612                                      * ビットを結び付けて
613      pea      (a2)                * 内容を整合させる
614      SXCALL    $A1D1              * __GMCalcGraph
615      addq.l    #4, sp
616
617      pea      (a2)                * 作成したグラフポートを
618      SXCALL    $A131              * カレントにする
619      addq.l    #4, sp              * __GMSetGraph
620
621      pea      winLocalRect(pc)    * 記録してあるスクリプトを
622      move.l    scriptHdl(a5), -(sp) * 描画
623      SXCALL    $A19C              * (テキストタイプで
624      addq.l    #8, sp              * 描画されるのがミソ)
625                                      * __GMDrawScript
626
627      move.l    $16(a1), d6
628      move.l    $0a(a1), a1
629      add.w     #4+4+8+4+4+32, d5
630
631      move.l    d5, -(sp)            * スクラップ用の
632                                      * セルリストの作成開始
633
634      SXCALL    $A021
635      addq.l    #4, sp
636
637      beq       Copy2Scrap9
638
639      move.l    d0, a4
640      move.l    (a4), a0
641      move.l    #'PAT4', (a0)+
642      addq.l    #8, d6
643      move.l    d6, (a0)+
644      clr.l     (a0)+
645      move.l    #WIN_X*$10000+WIN_Y, (a0)+
646      subq     #8, d6
647      subq     #1, d6
648      Copy2Scrap0:
649      move.b    (a1)+, (a0)+
650      dbra     d6, Copy2Scrap0
651      move.l    #'PAL2', (a0)+
652      move.l    #32, (a0)+
653      lea      paletData(pc), a1

```

* 最初のセルは
* 'PAT4' タイプ
* 'PAT4' サイズ
* バウンドレクタングル
* ビットの内容をコピー
* 非効率きわまりない
* 次のセルは
* 'PAL2' タイプ
* 'PAL2' サイズ


```

654      moveq    #16-1, d6
655 Copy2Scrap1:
656      move.w   (a1)+, (a0)+      # パレットデータをコピー
657      dbra     d6, Copy2Scrap1
658
659      pea      (a4)              # セルリストを収めたブロック
660      move.l   d5, -(sp)         # セルリストのサイズ
661      SXCALL   $A390             # TSPutScrap
662      addq.l   #8, sp            # でスクラップに送る
663
664      pea      (a4)              # セルリストのブロックを
665      SXCALL   $A038             # MMHdlDispose
666      addq.l   #4, sp            # で廃棄
667 Copy2Scrap9:
668      move.l   bitsHdl(a5), -(sp) # ビッツを
669      SXCALL   $A1CB             # _GMDisposeBits
670      addq.l   #4, sp            # で廃棄
671
672      move.l   a3, (a2)          # 一応もとの
673      pea      (a2)              # ビットマップに戻して
674      SXCALL   $A12E             # グラフポートをクローズ
675      addq.l   #4, sp            # _GMCloseGraph
676
677      SXCALL   $A0B8             # 通常のマウスポインタへ
678      # _EMDeCross
679
680      unlk     a6
681
682      rts
683
684 .even
685 winTitle:
686      dc.b     12, ' グラフSAMPLE' # ウィンドウタイトル
687 .even
688 winLocalRect:
689      dc.w     0, 0, WIN_X, WIN_Y   # ウィンドウの大きさを
690      # ローカル座標で
691      # 示すレクタングル
692      dc.w     8, 8, 192, 192      # 円グラフの大きさ
693
694 cTitle:
695      dc.b     16, ' クリップボードへ' # ボタンのタイトル
696      .even
697 cRect:
698      dc.w     200, 178, 316, 198    # ボタンの位置と大きさ
699      .even
700 paletData:
701      RGB      0, 0, 31, 0          # パレットデータ
702      RGB      31, 0, 0, 0          # 青
703      RGB      0, 31, 0, 0          # 赤
704      RGB      31, 0, 31, 0         # 緑
705      RGB      0, 31, 31, 0         # 紫
706      RGB      31, 31, 0, 0         # 水色
707      RGB      0, 0, 16, 0          # 黄色
708      RGB      16, 0, 0, 0          # 暗い青
709      RGB      0, 16, 0, 0          # 暗い赤
710      RGB      16, 0, 16, 0         # 暗い緑
711      RGB      0, 16, 16, 0         # 暗い紫
712      RGB      16, 16, 0, 0         # 暗い水色
713      RGB      0, 31, 24, 0         # 暗い黄色
714      RGB      0, 20, 31, 0         # ベーバーミントグリーン
715      RGB      31, 20, 24, 0         # (趣味)
716      RGB      0, 20, 31, 0         # ライトブルー
717      RGB      31, 20, 24, 0         # 謎の色

```

```

715          RGB      31, 31, 31, 0          # 白
716
717 inpPtr:
718          dc. b      255                    # 255文字まで読み込める
719          ds. b      1+256                  # 1行入力バッファ
720
721 fileErrMsg:
722          dc. b      'ファイルが読み込めません。', 0
723
724 formErrMsg:
725          dc. b      'グラフデータではありません。', 0
726
727          .end

```

■リスト 4 GRPSMPL 用 makefile

```

# GRPSMPL用 makefile
GRPSMPL. X:      SKELTON. o GRPSMPL. o
               lk -oGRPSMPL SKELTON GRPSMPL

SKELTON. o:      SKELTON. s WORK. INC
               as SKELTON

GRPSMPL. o:      GRPSMPL. s WORK. INC
               as GRPSMPL

```

第 3 章

新設されたマネージャ

SX-WINDOW バージョン 1.10 で大きく変わった点の 1 つは、既存のマネージャの拡張/改良だったわけですが、そればかりではなく、新たに 2 つのマネージャが追加されています。この 2 つのマネージャは、いずれも以前の SX-WINDOW では実現できなかったような新しいアプリケーションに道を拓いてくれるはずです。

3¹ | プリントマネージャ

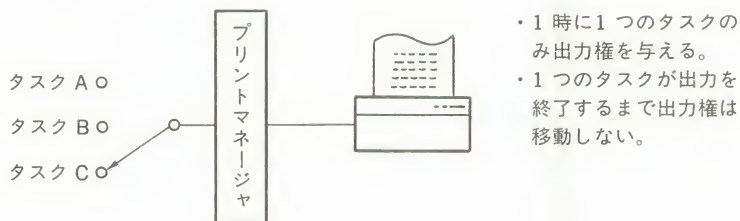
SX-WINDOW の OS としての役割の 1 つとして、資源を管理するという仕事があります。主記憶や補助記憶装置、CPU 時間のほかにも、X68000 に付属しているさまざまな装置を管理しなければならないのですが、バージョン 1.10 以前の SX-WINDOW では、ごく基本的なプリンタの管理が不十分でした。プリンタへの出力を管理するソフトウェアはリソースの中に用意されていましたが、その機能はテキストデータの出力と画面全体のハードコピー程度であるうえ、複雑な手順を必要としました。

バージョン 1.10 では、プリンタ出力に関する機能を SX-SYSTEM の中に統合し、SX コールによってかんたんに呼び出せるようになっています。この機能を司っているのがプリントマネージャです。

1 | プリントマネージャの機能の概要

X68000 はセントロニクス準拠のプリンタポートを 1 つ備えているので、原則として接続できるプリンタの数は一度に 1 台です。これに対して、SX-WINDOW 上で動作するタスクは複数であり、複数のタスクが無秩序にプリンタポートへデータを出力した場合、正常な印刷結果を望むことはできません。プリントマネージャの機能の第 1 は、プリンタポートへの出力権を管理し、タスク間で競合が発生しないように調停することにあります。プリントマネージャを正しく利用しているかぎり、ほかのタスクが印刷を行っているときには、ほかのタスクは印刷を開始できないので、プリンタポートの奪い合いは発生しません (図 1)。

■図 1 プリンタポートへの出力権の調停



第 2 の機能は、プリンタの機種間の差異を吸収し、決められた手順にさえしたかえれば各種のプリンタで同一の印刷結果が得られることです。Human では X68000 にさまざまな種類のプリンタが接続できるように、プリンタの機種の違いをプリンタデバイスドライバ(prndrv.sys 等)で吸収してきました。SX-WINDOW でもその思想は継承され、プリンタドライバ*1を用意してプリントマネージャに登録することでさまざまなプリンタに対応することができるようになっています*2。アプリケーションからの印刷要求は、プリントマネージャを通じて

プリンタドライバに送られ、コントロールパネル等で指定したプリンタドライバ内の印刷ルーチンによって実際の印刷が行われます。

- *1: プリントマネージャに登録するプリンタドライバと、Human の config. sys に登録するプリンタ用デバイスドライバとは別のものです。Human のプリンタ用デバイスドライバは *.sys のファイルとして用意され、config. sys の DEVICE=行によって登録しますが、プリントマネージャのプリンタドライバはリソース PRTD として用意され、コントロールパネル等で登録します。
- *2: バージョン 1.02 でも同様の形式でプリンタドライバは存在しましたが、それはバージョン 1.10 のもののほど完成されたものではありませんでした。その機能としては、デスクトップ全体のハードコピー、そして印刷する文字コードをバッファリングして Human のプリンタ用デバイスドライバへの引き渡し、といったものでした。バージョン 1.10 のプリンタドライバは、Human のデバイスドライバとは完全に独立した存在です。そのため、ほとんど意味はありませんが、Human と SX-WINDOW でプリンタを使い分けるといった芸当も可能です。

以上に加えて、プリントマネージャには図形や文書の印刷に便利なユーティリティが含まれています。

プリントマネージャを使って行う印刷には、大きく分けて、「コード印刷」、「ページ印刷」、「プロセス印刷」、そして「例外的な印刷」の4つを挙げることができます。それぞれについて解説します。

コード印刷

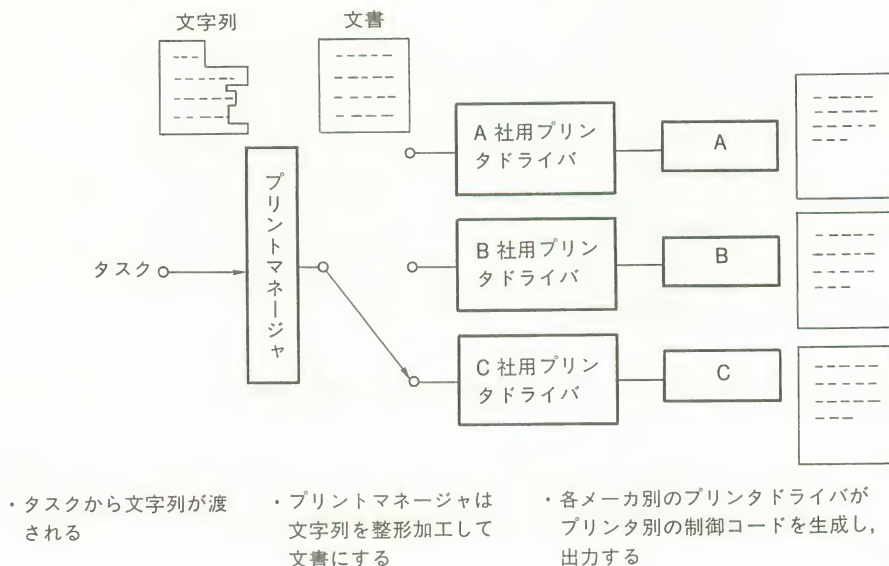
コード印刷は、文字のみで構成される文書データを印刷するための機能です。

一般に、文字を印刷するためには ASCII コード（あるいはカナ、カナ記号を含めて JIS コード）をプリンタに出力するだけで事足ります。ただし、漢字などの全角文字を印刷しようと思った場合は少々面倒です。日本のパソコンに接続して使うプリンタのほとんどには漢字フォントが搭載されているので、JIS 漢字コードと制御コードを出力することで全角文字を印刷することができます*3。しかし、プリンタの機種系列ごとに制御コードが異なっているので、使用しているプリンタの機種に適合したプリンタドライバを登録しておく必要があります。つまり、プリンタドライバの登録さえ行っておけば、プリンタ機種間の相違は吸収され、どのような環境で動作している SX-WINDOW 上でも、ほぼ同様な文書の印刷結果が得られるのです（172 ページ図 2）。

- *3: 外字などプリンタがフォントを持っていない文字をビットイメージとして印字する場合もあります。

ところで、Human や MS-DOS のプリンタデバイスを利用してテキストファイルの印刷を行った場合、プリンタの紙の種類によらず、つねに印刷形式は一定でした。横 80 文字（半角換算）、縦方向はとくに決まっていないというのが普通だったわけですが*4、これは、連続用紙を使用するラインプリンタが主流であった時代の名残りでした。現在使われているプリンタの多くは、A4 判をはじめとする多様なサイズの普通紙（あるいは感熱紙）を使うように（あるいは使えるように）なっており、実際に印刷の大部分はこうした用紙に行われているよう

■図2 コード印刷のイメージ



です。Human や MS-DOS のプリンタデバイスの過去を引きずった印刷形式というのは、こうした現状に即しているとはいえません。

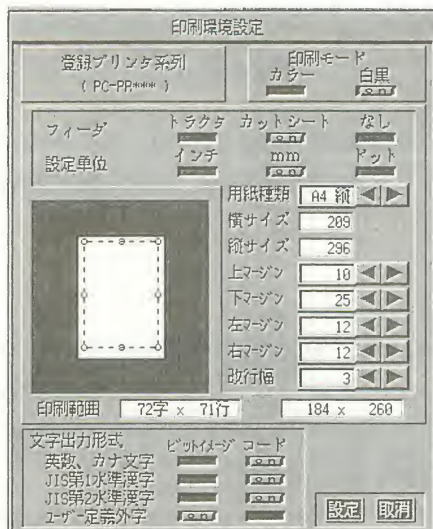
*4: Human のプリンタデバイスの場合は、デバイスドライバ登録時にオプションを指定することによって1ページの横文字数や行数を指定することが可能であるなどの配慮をうかがうことができます。これがDOS系の機械になると、こういった心遣いは皆無です。DOS系の機械の場合は、アプリケーションがそれぞれ印刷機能を持っているので不自由を感じないのかもしれませんが、すべてのアプリケーションに共通な印刷環境という思想が欠落しているために、ユーザはさまざまな不利益をこうむることになります。

また、左右上下マージンや行間隔、文字間隔を設定できないなど、ワープロの出力を見慣れた、目の肥えたユーザから見れば、「使いものにならない」と思われてもしかたがないかもしれません。

プリントマネージャでは、コード印刷にはじめからレイアウトの概念を導入して、プリンタの機能や使用する用紙、そして印刷する目的に即したレイアウトで印刷を行うことができるようになっています。こうしたレイアウトをはじめとする印刷環境は、コントロールパネルなどでかんたんに設定することが可能です(173ページ図3)。

印刷環境が設定されてしまえば、アプリケーションは印刷環境を意識する必要がなくなります。コード印刷を行いたい文字列(文書)を渡すだけで、プリントマネージャは設定された印刷環境という「型」に文字列を流し込んで印刷を行ってくれるのです。

■図3 印刷環境設定ダイアログ

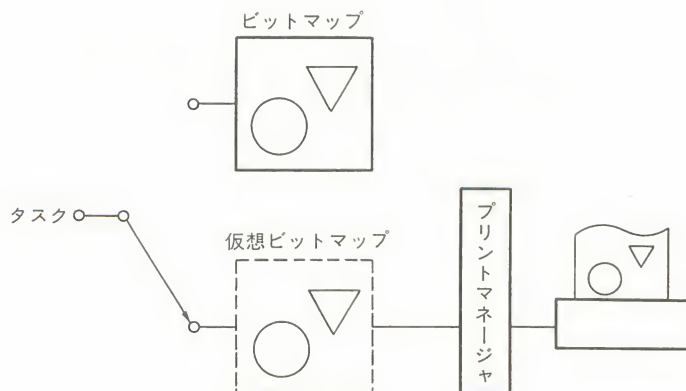


ページ印刷

ページ印刷は、主として画像や図形などを印刷するための機能です。

ページ印刷では、メモリ上に仮想のグラフィポートを作成し、そこに描画したものがプリンタ用紙の1ページとして印刷されます。つまり、グラフィックマネージャを使って描画できるものならば、ほとんどそのままプリンタに印刷することが可能なのです(図4)。ただし、内部ではスクリプトを利用して処理が行われているため、スクリプトに記録されるSXコールのみ使用可能です。

■図4 ページ印刷のイメージ



このグラフポートとビットマップはディスプレイ装置とは無関係なので、画面への描画とプリンタへの印刷を完全に別扱いにすることができます。従来、プリンタに印刷するためだけに画面に描画を行い、それをハードコピーしていたことを考えると、はるかにスマートな方法であることはおわかりいただけると思います。

コード印刷同様、プリンタの機種間の差異はプリントマネージャ、プリンタドライバによって吸収されるので、どのような環境で動作している SX-WINDOW 上でも、ほぼ同様な印刷結果が得られます。

プロセス印刷

プロセス印刷はページ印刷と同様、おもに画像や図形を印刷するための機能です。

ページ印刷では、スクリプトに記録できる SX コールによって描画されたものだけが印刷可能でしたが、プロセス印刷ではグラフィックマネージャによる描画に加えて、直接ビットマップを操作して、その結果を印刷することが可能です。

描画を行うサブルーチンはユーザープロセスと呼ばれ、プロセス印刷の準備の段階でプリントマネージャに登録しておくことで、描画/印刷が実行されることになります。

プロセス印刷の応用例として、カラー画像の印刷が考えられます。カラーの画像を白黒のプリンタで印刷するには、色の明暗をタイリングで表現しなければなりません。こういった芸の細かい作業はグラフィックマネージャで行おうとすると、かなり手間がかかります。このような場合、元のカラー画像を白黒のタイリング表現に変換して直接ビットマップに書き込むようなユーザープロセスを用意したプロセス印刷が適しています。

例外的な印刷

SX1.02 以前では、プリントマネージャは SX-SYSTEM 内に収められておらず、リソース PRTM の形^{*5}で提供されてきました。その名残りとして、SX1.10 でも PRTM は用意されていますが、プリントマネージャが SX-SYSTEM に統合されたことにより、ユーザーティビ的な使われ方が主となっています。

^{*5}: リソースタイプ PRTM, ID0. SYSTEM.LB に含まれています。前著『SX-WINDOW〜』222 ページ参照（ただし、この時点での表記は「プリンタマネージャ」）。

SX1.10 の PRTM のおもな機能は次の 3 つです。

- 1) テキストファイルの印刷
- 2) ファイルのダンプ出力
- 3) デスクトップ全体のハードコピー

これらの使い方等については、『SX-WINDOW〜』222 ページを参照してください。

2 印刷の仕組み

プリントマネージャの内部で、どのようにして印刷が行われているかを解説します。

印刷環境レコード

プリントマネージャが印刷を行うための各種情報は、印刷環境レコードにまとめられています。印刷環境レコードは、\$8B バイトにおよぶ大きなレコードです。各種印刷を行う前に、アプリケーションは印刷環境レコードとなる再配置可能なメモリブロックを作成し、その中に必要な情報をセットしておかなければなりません。

印刷環境レコードの内容は以下のとおりです。

オフセット	欄 名	内 容	
+\$00.w	prPaperKind	用紙の種類	(1)
+\$02.w	prPaperOption	プリンタオプションの有無	(2)
+\$04	prPaperRect	用紙のサイズを示すレクタングル	(3)
+\$0c	prLimitRect	印刷可能な範囲を示すレクタングル	(4)
+\$12	prPageRect	実際に印刷を行う範囲を示すレクタングル	(5)
+\$1c	prPaperRsv	システム予約 (8 バイト)	
+\$24.w	prDocImage	ビットイメージ出力フラグ	(6)
+\$26.w	prDocColumn	1 行の文字数	(7)
+\$28.w	prDocLine	1 ページの行数	(8)
+\$2a.w	prDocTab	タブサイズ	(9)
+\$2c.w	prDocHeight	改行幅	(10)
+\$2e	prDocRsv	システム予約 (8 バイト)	
+\$36.l	prRes	縦横の解像度を示すポイント	(11)
+\$3a.l	prANKSize	半角文字の縦横のドット数を示すポイント	(12)
+\$3e.l	prKanjiSize	全角文字の縦横のドット数を示すポイント	(13)
+\$42.w	prColorKind	カラー印刷の色種類	(14)
+\$44	prPrnRsv	システム予約 (8 バイト)	
+\$4c.w	prManVer	プリントマネージャのバージョン	(15)
+\$4e	prManRsv	システム予約 (8 バイト)	
+\$56.w	prDrvVer	プリンタドライバのバージョン	(16)
+\$58	prDrvRsv	システム予約 (8 バイト)	(17)
+\$60.w	prMinPage	印刷範囲開始ページ	(18)
+\$62.w	prMaxPage	印刷範囲終了ページ	(19)
+\$64.l	prUserData	ユーザ用のデータ	(20)
+\$68	prUserRsv	ユーザ用の予約領域 (8 バイト)	(21)
+\$70.w	prFstPage	印刷開始ページ	(22)
+\$72.w	prLstPage	印刷終了ページ	(23)
+\$74.w	prDupPage	1 ページあたりの印刷枚数	
+\$76.w	prMode	印刷モード	(24)
+\$78.w	prMask	印刷モードのマスク	(25)
+\$7a	prJobRsv	システム予約 (8 バイト)	
+\$82.w	prPageCount	現在印刷中のページ	(26)
+\$84.w	prDupCount	現在印刷中の部数	(27)
+\$86	prWorkRsv	システム予約 (8 バイト)	

(1) 用紙の種類

プリンタで使用する用紙の種類を意味しています。

用紙と数値は次のように対応しています。

0	フリーサイズ
1	A3 用紙縦置き
2	A3 用紙横置き
3	A4 用紙縦置き
4	A4 用紙横置き
5	A5 用紙縦置き
6	A5 用紙横置き
7	B3 用紙縦置き
8	B3 用紙横置き
9	B4 用紙縦置き
10	B4 用紙横置き
11	B5 用紙縦置き
12	B5 用紙横置き
13	10×11 インチ連続用紙
14	15×11 インチ連続用紙
15	ハガキ縦置き
16	ハガキ横置き

(2) プリンタオプションの有無

プリンタに付属する周辺機器の種類を意味しています。

0	周辺機器なし
1	トラクタフィーダ付き
2	カットシートフィーダ付き
3	ハガキフィーダ付き

(3) 用紙のサイズを示すレクタングル

(4) 印刷可能な範囲を示すレクタングル

(5) 実際に印刷を行う範囲を示すレクタングル

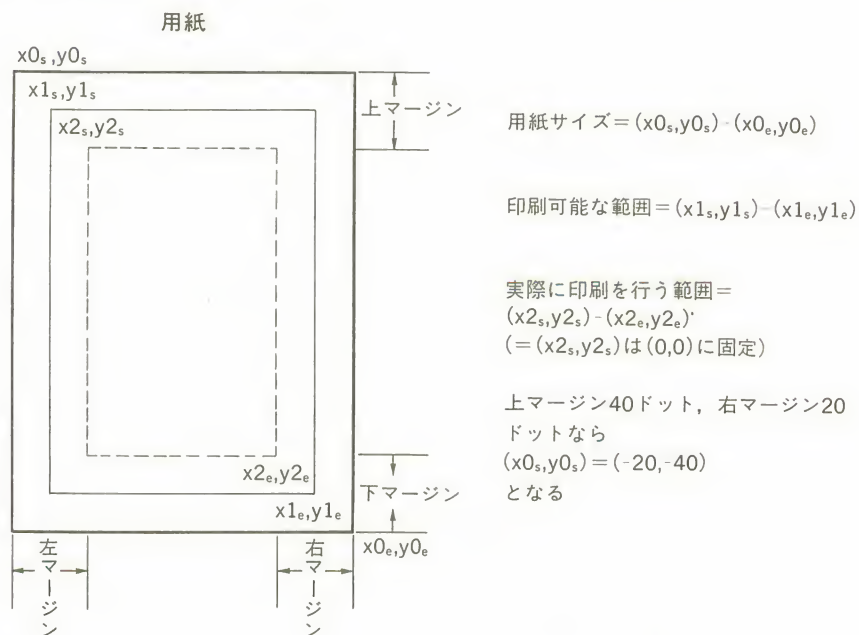
これらのレクタングルは 1 ページ (1 枚のカット紙、ハガキ、あるいは連続用紙の 1 ページ) 中の印刷を行う範囲を示しています。

(5) の実際に印刷を行う範囲がホーム位置となり、用紙のサイズや印刷可能な範囲の左上の座標は負の値となります (177 ページ図 5)。

(6) ビットイメージ出力フラグ

コード印刷を行う際に、プリンタの内蔵フォントを使わずにビットイメージで印字する文字の種類を意味します。1 ワード中の各ビットが文字の種類に対応し、1 になっている文字がビットイメージで印字されます。

■図 5 各レクタングルの関係



bit0	外字
bit1	システム予約
bit2	全角文字・JIS 第2水準
bit3	全角文字・JIS 第1水準
bit4	半角文字

- (7) 1行の文字数
- (8) 1ページの行数
- (9) タブサイズ
- (10) 改行幅

コード印刷時の、1ページに印刷される文書のおおまかな書式が収められています。1行の文字数、1ページの行数、タブサイズは半角文字を単位として、改行幅はドットを単位として指定します (178 ページ図 6)。

(11) 縦横の解像度を示すポイント

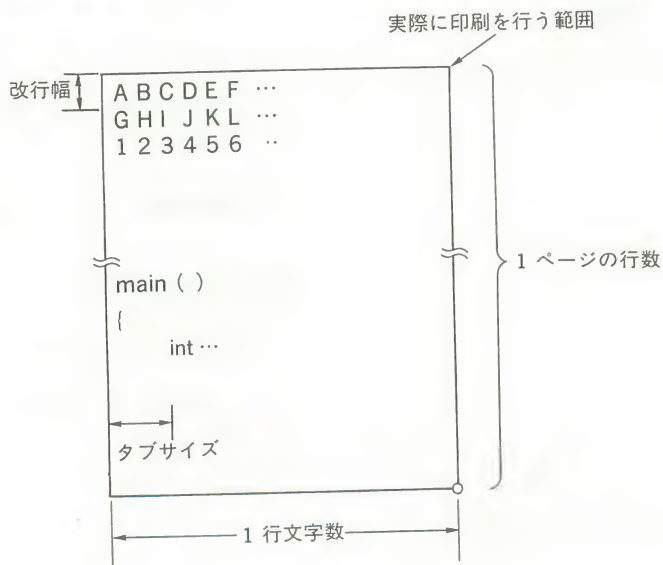
プリンタの横方向、縦方向の解像度を DPI (ドット/インチ) 単位で示します。この数値はポイント形式で、上位ワードは横方向、下位ワードは縦方向の解像度を意味しています。

(12) 半角文字の縦横のドット数を示すポイント

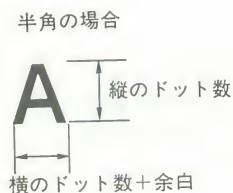
(13) 全角文字の縦横のドット数を示すポイント

半角、全角文字の縦横のドット数をポイント形式で示します。すなわち、上位ワードが横方向のドット数、下位ワードが縦方向のドット数です (178 ページ図 7)。実際に文字を構成する

■図6 書式関係の数値の意味



■図7 半角，全角文字の縦横のドット数



ドットの数だけではなく，文字数調整用の余白も含めて指定します。

(14) カラー印刷の色種類

プロセス印刷時の色の表現のしかたを意味しています。

(15) プリントマネージャのバージョン

プリントマネージャのバージョンを意味します。SX1.10 のプリントマネージャのバージョンは 1.00 なので，ここには\$0100 という数値が入るのが普通です。

(16) プリンタドライバのバージョン

プリンタドライバのバージョンを意味します。SX1.10 に付属するプリンタドライバのバージョンはいずれも 1.00 なので，ここには\$0100 という数値が入るのが普通です。

(17) 印刷範囲開始ページ

(18) 印刷範囲終了ページ

現在は使用されていません。

(19) ユーザ用のデータ

(20) ユーザ用の予約領域

アプリケーションが自由に使用することができます。

(21) 印刷開始ページ**(22) 印刷終了ページ**

印刷を開始する前にアプリケーションが指定します。

ページは 1 ページから始まり、デフォルトでは 1~999 ページまでを印刷するように設定されています。

ページ印刷の場合は、任意のページ番号をセットすることで正しく印刷の開始/終了が行われますが、コード印刷の場合、印刷開始ページはデフォルトの 1 から変更してはいけません。

(23) 1 ページあたりの印刷枚数

1 ページを何部印刷するかを、印刷を開始する前にアプリケーションが指定します。

(24) 印刷モード

印刷のモードを指定します。1 ワードの各ビットが意味を持ち、1 になっている印刷モードが有効となります。印刷を開始する前にアプリケーションが指定します。

bit0	ドラフト印刷（ドットを間引く）
bit1	カラー印刷

(25) 印刷モードのマスク

プリンタの種類によっては印刷モードを制限する必要があります。このマスクと印刷モードを AND したものが実際の印刷モードとなります。

この値はプリンタドライバが設定するので、変更してはいけません。

(26) 現在印刷中のページ**(27) 現在印刷中の部数**

現在印字中のページに関する情報がプリンタドライバによってセットされます。

以上のような情報は、アプリケーションが自分で用意した値によって設定することもできます。しかし、場合によっては、たかひに矛盾する情報を設定してしまうことも考えられます*6。印刷環境レコード内の情報ができるだけ正確になるように、また、そのためのアプリケーションの負担をできるだけ軽減するように、プリントマネージャにはいくつかの便利な機能が用意されています。

*6：用紙をはみだすような 1 行の文字数や、異常なレクタングルの数値など。

(a) 印刷環境レコードの内容をチェック/調整する機能

印刷環境レコード内の情報が矛盾していないかどうかチェックし、不都合を発見した場合はその値を調整する機能が、\$A4E5 PMValidate として用意されています。アプリケーションが値を設定した場合は、この SX コールによって、その正しさをチェックするのが安全です。

(b) デフォルトの印刷環境レコードの情報をセットする機能

プリンタドライバ内部には、印刷環境レコード内に収めるデフォルトの情報が用意されています*7。\$A4E4 PMSetDefault によって、この機能を利用することができます。あらかじめ、この SX コールでデフォルトの値をセットしておいて、アプリケーションは必要とところだけを書き換えることで労力を減らすことができます。

*7:コントロールパネルで設定した情報が BUILTIN.LB 内のリソースタイプ PrEV, ID0 として記録されている場合は、そちらを優先的に読み込みます。

(c) 印刷環境設定ダイアログによって情報をセットする機能

印刷環境設定ダイアログとは、173 ページの図 3 に示したようなダイアログです。このダイアログを利用するためには、\$A4E6 PMImageDialog*8 を呼び出すだけでよく、後の処理はプリントマネージャとプリンタドライバが行ってくれます。

*8:\$A4E7 PMStrDialog という、コード印刷専用の印刷環境設定も用意されていますが、SX1.10 に付属してきたプリンタドライバは、いずれもこの機能をサポートしていません。

本来、この SX コールではページ印刷時の印刷環境を設定するためのダイアログが表示され、その結果が印刷環境レコードのページ印刷に関係する値だけに反映されることになっているようですが、実際にはコード印刷の印刷環境設定も兼ねています。

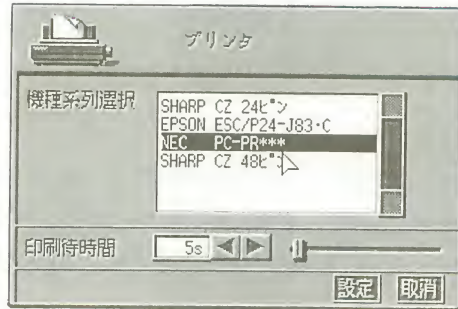
プリンタドライバ

プリンタに印刷を行うのがプリントマネージャの仕事であると思ってしまうがちですが、厳密にはそうではありません。プリントマネージャの仕事はプリンタに印刷を行うためのデータの流れを管理することであって、プリンタポートへのデータの出力などはいっさい行っていません。そういった下位レベルの作業は、プリンタの機種ごとに用意されたプリンタドライバが行っています。

プリンタドライバはリソースのかたちで提供されます。SX1.10 には「CZ シリーズ 24 ピン系」、「ESC/P 系」、「PG-PR 系」、そして「CZ シリーズ 48 ピン系」の 4 つのプリンタドライバが提供されており、それぞれリソース PRTD の ID0, 1, 2, 3 として SYSTEM.LB に収められています。これらを「目にする」数少ない機会としては、コントロールパネルでプリンタの設定を行う場合が挙げられます (181 ページ図 8)。

プリンタドライバの実態は一種のプログラムモジュールで、形式としては R 型のモジュールに近いかたちになっています。しかし、いわゆるリソースからのタスクの起動というかたちではなく、リソースをメモリに読み込んで、読み込まれたアドレスをコールするという、ウィンドウマネージャのウィンドウ定義関数に近いかたちでプリンタドライバから利用されています。このため、ウィンドウ定義関数同様、リロケータブルでリエントラントなコードでなければなりません。

■図 8 プリンタドライバ (コントロール.Xより)



プリンタドライバの機能の詳細や作成時の注意などについては、198 ページの「4 プリンタドライバの作成」で述べることにして、ここではプリントマネージャがどのようにプリンタドライバを利用し、印刷を行っているかを解説することになります。

もっともシンプルな例として、コード印刷の場合を取り上げることになります。

すでに述べたように、英数字だけで構成された文書であれば、プリンタの機種にはおおむね関係なく、単純に ASCII コードを流せば、とりあえず印刷することだけはできます。コード印刷では、全角文字の混じった文書を扱うことはもちろん、ここにレイアウトの概念を導入したために、より高度にプリンタをコントロールする必要が出てきました。

プリンタをコントロールする制御シーケンスは、プリンタの機種によって異なっています。全角文字を打ち出すだけでも、漢字モードに切り替え、JIS コードを出力し、漢字モードを終了するという手順が必要なのわけですが、漢字モードへの切り替え、漢字モード終了を指示する制御シーケンスからしてもうバラバラです。まして、改行幅や文字間隔の制御やビットイメージの印刷などにいたっては、どういう状況かは想像に難くないでしょう。

プリントマネージャは、アプリケーションからコード印刷の要求とともに印刷すべき文書(=文字列)を受け取ると、ユーザが自分の環境にあわせて選択しておいたプリンタドライバをリソースから呼び出し、文字列を渡し、印刷開始を指示します。プリンタドライバは、文字列をもとに、印刷環境レコードにしたがって文字間隔などを調整しつつ、印刷を行います。

ページ印刷やプロセス印刷の場合も同様に、プリントマネージャからは比較的抽象的なデータが渡され、プリンタドライバはそれぞれのプリンタに実際に印刷できるかたちに変換し、出力します。

実際に印刷を行う機能のほかに、プリンタドライバには印刷環境レコードの設定(デフォルト設定、チェック&調整、印刷環境設定ダイアログの表示&コントロール)などのユーティリティ的な機能も含まれています。

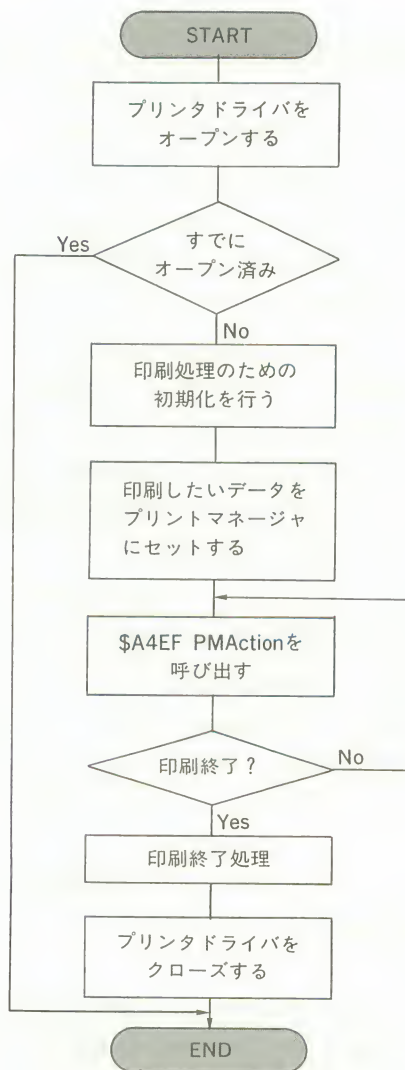
これらのプリンタドライバの仕事は、アプリケーション側から見ると、プリントマネージャの仕事であるように思えます。実際に、プリンタドライバの登録さえすんでいるのなら、アプリケーションはプリンタドライバの存在を意識することなく、プリントマネージャの SX コー

ルを利用するだけで印刷を行うことができます。しかし、いずれも実際の作業はプリンタドライバが行っていて、プリントマネージャはその間を取り持つのが仕事であることは、覚えておいて損はないと思います。

基本的な印刷の流れ

プリントマネージャを利用した印刷は、一定の手順を経て行います。コード印刷の場合、ページ印刷の場合、プロセス印刷の場合など、それぞれに少しずつ異なっていますが、基本的な流れはどれも共通しています。これをフローチャートで表すと、図9のようになります。

■図9 印刷の流れ



最初にプリンタドライバをオープンする際、すでにオープンされているかどうかチェックするのは、タスク間で印刷作業が競合しないようにするためです。すでにオープンされている場合は、ほかのタスクで印刷を行っているということですから、印刷作業は行うことができません。エラー用のダイアログなどでユーザに警告を出して、印刷ができない旨を伝えて印刷処理を中断します。

「印刷したいデータをプリントマネージャにセットする」の部分は、印刷の種類によって異なります。コード印刷であれば、文書（文字列）の収められたメモリブロックへのハンドルをプリントマネージャに渡します。ページ印刷ならば、印刷用の仮想のグラフポートを開いて、そこに描画を行うこととなります。プロセス印刷の場合は、データをセットするかわりにユーザープロセスをプリントマネージャに登録します。

ここまでは印刷の準備の段階で、実際の印刷はまだ行われていません。

プリンタへの出力は、その次の「\$A4EF PMAAction を呼び出す」*9を実行するたびに少しずつ行われます。少しずつ、ということは、\$A4EF PMAAction を何度か呼び出さなければ、すべての必要なデータの印刷は行われないということです。不便に思われるかもしれませんが、これはタスクマネージャによるマルチタスクを利用して印刷を行うための合理的な仕様です。

*9:\$A4EF PMAAction は 1 ワードの引数を取ります。これによって、印刷作業の中断や再開などを指示できるのですが、ここでは「印刷の続行」を意味する 0 を指定していると理解してください。

プリンタは MPU から見ると非常に遅いデバイスなので、すべての CPU 時間を費やしてデータを最後まで印刷していたのでは時間がかかりすぎるうえ、ほかのタスクも止まってしまいます。割り込みを利用して完全にバックグラウンドで処理を行うことも可能ではありますが、ほかのマネージャとの関係や、エラー処理などの点で問題があります。

そこで、印刷の作業を十分短い時間で実行できるように細かく分割して、適当な間隔をおいてそれを実行することによって、ほかのタスクに大きな影響を与えることなく、またシステム全体に破綻をきたすこともなく、マルチタスクを生かした印刷が可能となります。これは、第 1 章で示した「リアルタイムで動作する（ように見える）プログラム」と同じ発想です。

このような仕組みを利用して効率よく印刷を行うためには、印刷したいデータをプリントマネージャにセットしたら、実際の印刷処理である \$A4EF PMAAction はアイドルイベントで呼び出すようにすべきでしょう。ほかのタスクを止めて印刷に専念したい場合もあると思いますが、その場合はマウスポインタを踏切ポインタにする等、ユーザがほかの操作ができないことを示すようにすべきです。

\$A4EF PMAAction がすべての印刷を終了したことを示す値を返してきたら、終了処理を行って、プリンタドライバをクローズします。「印刷の終了処理」はページ印刷でのみ必要な処理ですが、プリンタドライバのクローズはすべての種類の印刷で行わなければなりません。プリンタドライバが占有するメモリの解放という意味ももちろんありますが、それと同時に、

ほかのタスクへのプリンタポートの利用権の移譲という意味も兼ねているからです。

ページ印刷の仕組み：スクリプトの利用

ページ印刷では、「印刷したいデータをプリントマネージャにセットする」方法としてスクリプトを利用しています。

ページ印刷用のデータのセットは、\$A4EB PMOpenImage で印刷用の仮想のグラフポートを作成することから始まります。このグラフポートはビットマップとはつながっておらず、スクリプトを記録するための形式的なものであると考えてください。

仮想のグラフポートが作成できたら、直後に\$A4EC PMRecordPage を呼んでスクリプトへの記録を開始します。

PMRecordPage を呼び出す際には、引数としてレクタングルレコードへのポインタを渡します。このレクタングルは、描画を行う範囲を意味しています。注意していただきたいのは、あくまでも「描画を行う範囲」であって、「印刷を行う範囲」ではない、ということです。あまり違わないようにも思えますが、じつは大きな違いです。この違いは印刷の時点で明らかになります。

この状態で、このグラフポートを通じて行う描画は、スクリプトに記録されます。いつもと同じように、グラフィックマネージャを利用して図形を描画してください。スクリプトへの記録中ですから、画面に表示されることはありません*10。こうして描いた図形は、ほぼそのままプリンタ用紙の上に印刷されることになります。

*10: もっとも、スクリプト記録中でなかったとしても、このグラフポートはビットマップとつながっていないので、バスエラーが発生するのがオチです。

必要な描画がすんだら、\$A4ED PMPrintPage を呼びます。この時点でスクリプトの記録は終了し、描画の手順を記録したスクリプトレコードが残ります。プリンタドライバは、このスクリプトレコードをもとに少しずつ仮想のビットマップに描画を行い、プリンタにあわせたデータ変換を行って出力することになります。

さて、この仮想のビットマップの大きさですが、印刷環境レコードに記録されている「実際に印刷を行う範囲」と等しいと考えてください*11。先ほど記録したスクリプトは、指定した描画範囲の中で記録されていたものですが、仮想のビットマップに描画される際には、このビットマップの大きさにあわせて拡大・縮小が行われます。ここで「描画を行う範囲」と「印刷を行う範囲」の違いが表れてきます。

*11: しかし、印刷を行う範囲が大きかった場合、その全体が収まるようなビットマップをメモリ中に作成すると、とてつもなく大きな領域が必要になることがあります。この対策は、次のプロセス印刷のところで明らかになります。

たとえば、描画範囲 (0, 0) - (256, 256) の状態で、(256, 0) - (0, 256) という、描画範囲いっぱいの対角線を引くようなスクリプトを記録したとします。それが、(0, 0) - (1440,

1440)のビットマップ上に再現された場合は拡大・縮小が³ほどこされ、(1440, 0)-(0, 1440)の、印刷範囲いっぱいの対角線として印刷されることになります。

つまりは、描画範囲いっぱいの描画を記録すれば、(プリンタ用紙いっぱいに印刷範囲が³設定されていた場合には)プリンタ用紙いっぱいの印刷が行われる、ということです。

プロセス印刷の仕組み：ユーザープロセス

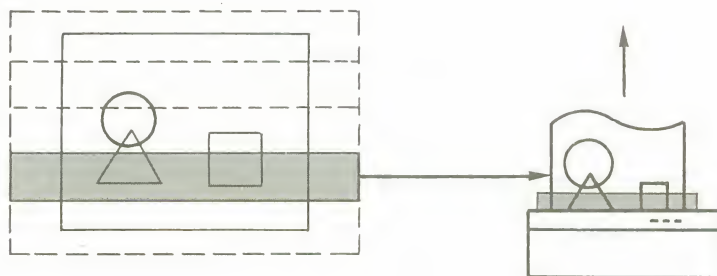
コード印刷やページ印刷は、「データを用意する」→「用意したデータが印刷される」という、わかりやすい印刷方法であるわけですが、プロセス印刷は多少趣きが異なります。

プロセス印刷では、印刷するデータは用意しません。そのかわり、印刷すべきデータを生成するルーチンを用意します。これがユーザープロセスです。

プロセス印刷で印刷が行われるのは、印刷環境レコードの中の「実際に印刷が行われる範囲を示すレクタングル」の内部です。これと同じ大きさの仮想ビットマップ/グラフポートが用意されるので、ユーザープロセスはグラフィックマネージャなり、独自の描画ルーチンなりで印刷したい画像や図形を描画します。ここに描画されたものが、そのまま紙の上に印刷されることになります。

ただし、実際にそういったビットマップをメモリ上に作成した場合、非常に多くのメモリが必要となる場合があります。このため、これをある大きさの「帯」に分割し、それを1本ずつ描画して印刷することを繰り返すことにします。こうしておけば、メモリ上のビットマップは1本の「帯」の分だけ用意しておけばすみます (図 10)。

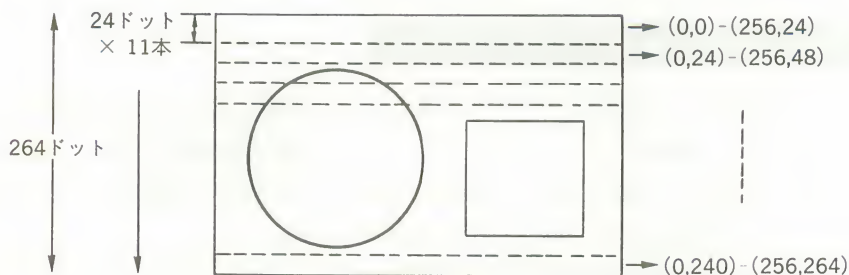
■図 10 プロセス印刷の仕組み



「帯」のサイズは、横方向は印刷を行う範囲と同じ、縦方向はプリンタの印刷ヘッドのピンの本数です*12。つまり、印刷する範囲が横 256 ドット、縦 264 ドットであった場合、24 ピンのプリンタ用ドライバのもとでは、1本の「帯」のサイズは横 256 ドット、縦 24 ドットということになり、全印刷範囲を印刷するためには、この「帯」を 11 本描画すればよいことになります (186 ページ図 11)。

*12:現在提供されているプリンタドライバはすべてシリアルプリンタ用なのでよいとして、将来ページプリンタ用のドライバが提供された場合、「ピンの本数」という表現は適当ではないかもしれません。

■図 11 ビットマップの分割



ユーザープロセスが呼び出されるのは、アプリケーションが\$A4EF PMAAction を呼び出したときです。このとき、ユーザープロセスにはスタック経由で次のような引数が渡されます。

long	prHdl	; 印刷環境レコードへのハンドル
long	frameRect	; 印刷範囲を意味するレクタングル

このほかに、呼び出された時点でのカレントグラフポートとして、仮想のビットマップへのグラフポートがセットされています。このビットマップ/グラフポートのビットマップレクタングルには「帯」のレクタングルがセットされているので、ここを調べることでユーザープロセスは描画範囲中のどの部分を描画することが求められているのかを知ることができます。

グラフィックマネージャや自前のルーチンで適切に描画を行い、それが終了したら、返り値として DO に 0 を入れて RTS します。ユーザープロセスが終了すると、プリントマネージャとプリンタドライバは、ビットマップの内容を各プリンタに適合したかたちに変換し、制御シーケンスを付加したうえで出力/印刷します。

1つの「帯」を出力し終わると、プリントマネージャはビットマップの位置を 1つ下の「帯」に移動させます。まだ描画/印刷すべき「帯」が残っている場合は「印刷中」の意味の値を、また、すべての「帯」を描画/印刷したと判断した場合は「印刷終了」の意味の値を持って、PMAAction を終了します。

先ほどの、24 ピンプリンタの、横 256 ドット、縦 264 ドットの印刷範囲の例であれば、ビットマップのレクタングルは次ページの表のように遷移することになります。

PMAction を 呼んだ回数	ビットマップのレクタングル
1 回目	(0, 0) - (256, 24)
2 回目	(0, 24) - (256, 48)
3 回目	(0, 48) - (256, 72)
⋮	
10 回目	(0, 216) - (256, 240)
11 回目	(0, 240) - (256, 264) ←これを印字し終われば終了
<終了>	

この例のように、印刷範囲の縦のドット数がピンの本数で割り切れる場合はよいのですが、端数が出る場合は少し注意する必要があります。たとえば、横 256 ドット、縦 265 ドットであった場合は、ビットマップのレクタングルは次の表のように遷移します。

PMAction を 呼んだ回数	ビットマップのレクタングル
1 回目	(0, 0) - (256, 24)
2 回目	(0, 24) - (256, 48)
3 回目	(0, 48) - (256, 72)
⋮	
10 回目	(0, 216) - (256, 240)
11 回目	(0, 240) - (256, 264)
12 回目	(0, 264) - (256, 288)
<終了>	

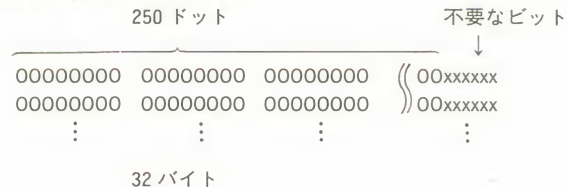
1 ドットはみだしたために、PMAction を 1 回多く呼ばなければならなくなっています。しかも、下線で示したように、12 回目の時点でのビットマップのレクタングルは、残っている 1 ドットよりも大きなものになっています。

このような場合のために、スタック経由で渡された印刷範囲のレクタングルをチェックして、必要な部分以外には描画を行わないようにする必要があります。

同様に、横方向のドット数が 16 の倍数でない場合、ビットマップの右端に不要なビットが生じます(図 12)。これらのビットをマスクする処理を行うのもユーザープロセスの責任です。

■図 12 ビットマップの右端の不要なビット

例：描画範囲の横が 250 ドットの場合



3 プリントマネージャの利用

プリントマネージャを利用して各種印刷を行う場合のコードの書き方について述べます。

ハードコピー以外については、182 ページの図 9 に示したフローチャートが基本となりますので、その流れを把握しておいてください。

コード印刷

コード印刷では、再配置可能ブロックの中に収められている文書（文字列）が印刷の対象となります*13。文書といっても、いわゆるテキストファイル形式で、文字コードと下の表に示した一部のコントロールコード以外は利用できないと考えたほうがよいでしょう。

*13:疑似ハンドルも使用可能です

コントロールコード	意 味
TAB (\$09)	水平タブ
FF (\$0C)	改ページ
CR (\$0D)	改行

ASCIIZ 型でも LASCII 型でもない、文書のバイト数を別に指定する形式の文字列ですから、文字列の終端を意味する\$00 や、文字列長を意味する先頭の 1 バイトなどは必要ありません。

コードの例は、こうした文字列が収められている再配置可能ブロックがすでに存在し、そのブロックへのハンドルがワーク strHdl に、文字列長がシンボル STRLEN に定義されていることを前提として示すことにします。

最初に行うのは、プリンタドライバのオープンです。これには\$A4E2 PMOpen を使います。リソース PRTD の ID 番号を指定することにより、どのプリンタ用のドライバをオープンするかを選択することが可能ですが、よほど特別な場合でないかぎり、オープンするドライバはコントロールパネルによって選択/登録されているものを利用することになります。この場合、ID 番号として-1 を指定します*14。

*14:コントロールパネルで選択したプリンタドライバの ID 番号は SRAM に記録されています。プリンタドライバのオープン時に -1 を指定した場合、SRAM を参照してプリンタドライバをオープンします。

CodePrint:

```

move.w    # -1, -(sp)
SXCALL    $A4E2                      * __PMOpen
addq.l    # 2, sp

```

このとき、返り値として -1 が返ってきた場合は、プリントマネージャでなんらかのエラーが発生しており、また -2 が返ってきた場合は、すでにプリンタドライバがオープンされていることを意味しています。いずれにせよ、負の数の場合はオープンに失敗しているので、印刷処理を中断します。

```
tst.l      d0
bmi        CodePrintAbort      *印刷処理を中断
```

次に、印刷環境レコードを作成します。印刷環境レコードはハンドルで指定するので、ヒープゾーンに再配置可能ブロックとして作成することになります。

```
move.l     #$8e,-(sp)           *印刷環境レコードのサイズ
SXCALL     $A021                * __MMChHdlNew
addq.l     #4,sp
move.l     d0,prHdl(a5)
```

印刷環境レコード内部の情報をセットします。まず、デフォルトの情報をセットしておき、必要があれば（ユーザからの要請があれば）、印刷環境設定ダイアログを表示します。ここでは、ユーザの要請によらず、かならず印刷環境設定ダイアログを出すことにします。

```
move.l     prHdl(a5),-(sp)
SXCALL     $A4E4                * __PMSetDefault
addq.l     #4,sp
move.l     prHdl(a5),-(sp)
SXCALL     $A4E6                * __PMImageDialog
addq.l     #4,sp
```

本来はコード印刷を行っているわけですから、\$A4E6 PMImageDialogではなく、\$A4E7 PMStrDialog を使うべきなのですが、注でも述べたように、現在提供されているプリンタドライバは PMStrDialog をサポートしていないので、PMImageDialog を使っています。

\$A4E6 PMImageDialog は、内容に変更があった場合は 1、変更がない場合は 0、エラーの場合は -1 を返します。変更されていた場合、それをリソースに保存しておきたいのなら、次のようなコードを書くといでしょう。

```
tst.l      d0                  *変更があった？
beq        CodePrint0         * なければ CodePrint0 へ
```

```

move.l    prHdl(a5),-(sp)
SXCALL    $A4F8                      * __PMSaveEnv
addq.l     # 4,sp

```

CodePrint0:

さらにレコードの内部に変更を加えたい場合は、直接操作してしまってもかまいません。しかし、その場合、設定に矛盾がないかどうか、チェックするために\$A4E5 PMValidate を呼ぶようにしてください。

印刷環境レコードの設定が完了したので、次は印刷する文字列をプリントマネージャに渡します。これには\$A4F1 PMDrawString を使います。

```

move.l     # 0, -(sp)                * 最後に改ページを行う
                                           * l を指定した場合は改ページしない
move.l     # STRLEN, -(sp)          * 文字列のバイト数
move.l     strHdl(a5), -(sp)        * 文字列へのハンドル
move.l     prHdl(a5), -(sp)        * 印刷環境レコードへのハンドル
SXCALL     $A4F1                    * __PMDrawString
lea        16(sp), sp

st         printActive(a5)          * 印刷中フラグを立てる

```

以上で準備は終了です。実際の印刷は、\$A4EF PMAction をアイドルイベントで呼び出すことで行います。アイドルイベントで PMAction を呼ぶべきかどうかを判断するためのフラグとして、変数 printActive を用意し、これが立っている場合は PMAction を呼ぶようにしています。

IDLE:

```

tst.b      printActive(a5)          * 印刷中?
beq        IDLE9                    * でなければ IDLE9 へ
move.w     # 0, -(sp)                * 「印刷続行」
SXCALL     $A4EF                    * __PMAction
addq.l     # 2, sp
tst.l      d0                      * 印刷終了?
beq        CodePrintFinish          * ならば CodePrintFinish へ

```

IDLE9:

```

moveq     # 0, d0
rts

```


印刷が終了したと判断できた場合は、CodePrintFinish でプリンタドライバをクローズして印刷処理のすべてを終了します。

CodePrintFinish:

```

        SXCALL    $A4E3                * __PMClose

        sf        printActive(a5)      * 印刷中フラグをおろす
        bra       IDLE9

```

印刷環境レコードを収めたメモリブロックを廃棄したりすることも忘れずに行ってください。

ページ印刷

コード印刷とページ印刷では、プリンタドライバのオープンから印刷環境レコードの設定まで、そして PMAction まわりはほとんど共通しているので、それ以外の部分について述べることにします。

ここでは印刷環境レコードの設定が終了した直後から始めます。

ページ印刷で最初に行うのは、\$A4EB PMOpenImage で仮想のグラフポートを作成することです。この結果、正常に作成できた場合は、返り値として A0 にグラフポートへのポインタが返ります。このポインタを直接利用することはあまりないと考えられますが³、いちおうワークに保存しておきます。

PagePrint:

```

                                :                (ここでドライバのオープン、印刷環境レコード
                                :                の設定などが行われる)
                                :
        move.l    prHdl(a5),-(sp)
        SXCALL    $A4EB                * __PMOpenImage
        addq.l    # 4,sp
        move.l    a0,graphPtr(a5)

```

この状態で、カレントグラフポートは、この仮想のグラフポートになっています。

続いて、スクリプトの記録を開始します。このとき指定する描画範囲のレクタングルへのポインタは、どこかにあるレクタングルレコード frameRect のアドレスを指定することにして、それ以上の条件は特定しないことにします。

```

        pea      frameRect            * 描画範囲を示すレクタングルへのポインタ
        SXCALL    $A4EC                * __PMRecordPage

```

```
addq.l    # 4,sp
```

以降、グラフィックマネージャの SX コールを呼び出して、描画を行います。

スクリプトに記録される SX コールであれば、ほとんどのものが利用できますが、ビットマップの変更とホーム位置の移動は行ってはいけません。

ここで描画を行う際は、次の条件のビットマップに描画するつもりで行ってください。

・テキストタイプ

描画するイメージ等はテキストタイプで行ってください。

・ページ数は 1 (白黒プリンタの場合)、または 3 (カラープリンタの場合)

ページ数 1 の場合は、カラー 0 が白、カラー 1 が黒として印刷されます。

ページ数 3 の場合は、描画色がプリンタのカラーコード 0~7 に対応します。

デフォルトの状態では、ペンの色、フォントの色は黒、背景が白となっています。

必要な描画が終了したら、スクリプトの記録を終了し、印刷データとしてプリントマネージャにセットします。これには \$A4ED PMPrintPage を使います。PMPrintPage はロングワードの引数を取りますが、かならず 0 を指定することになっています。

```
clr.l      -(sp)
SXCALL     $A4ED                                * __PMPrintPage
addq.l     # 4,sp
```

以上で印刷すべきデータのセットは完了です。後はコード印刷と同様、アイドルイベントで PMAction を呼び続けることで実際の印刷を行うことができます。

ただ一点、コード印刷と異なるのは、プリンタドライバをクローズする前に、\$A4F0 PMCcloseImage を使って仮想のグラフポートなどを廃棄する必要があります。

CodePrintFinish:

```
        SXCALL     $A4F0                                * __PMCcloseImage

        SXCALL     $A4E3                                * __PMCclose

        sf         printActive(a5)                    * 印刷中フラグをおろす
        bra        IDLE9
```

プロセス印刷

プロセス印刷を行う手順は、ほかの印刷とそれほど変わるものではありません。

印刷の準備では、ドライバのオープン、印刷環境レコードの設定はほかの印刷と共通です。

ほかの印刷がデータをセットするところで、プロセス印刷ではユーザープロセスの登録を行います。

ProcPrint:

```

:                                     (ここでドライバのオープン、印刷環境レコー
:                                     ドの設定などが行われる)
:
    pea      userProc(pc)            *ユーザープロセスへのポインタ
    move.l   prHdl(a5),-(sp)
    SXCALL   $A4FA                    * _PMProcPrint
    addq.l   #8,sp

```

以上で準備は終了です。後はコード印刷と同様にアイドルイベントで PMAction を呼び、終了したら、ドライバをクローズします。

問題は、どのようにユーザープロセスを書くか、です。

ユーザープロセスでは、「プロセス印刷の仕組み：ユーザープロセス」で述べたような処理を行うわけですが、ここでは例として、印刷範囲を 1 ラインずつ \$00, \$01, …… というビットパターンで埋めてみることにします。

ユーザープロセスで描画を行うビットマップは、ページ印刷同様 1 ページ、あるいは 3 ページのテキストタイプです。ここでは話をかんたんにするために、1 ページであることを想定しています。この場合、ビットマップ中の 1 になっているビットの部分が黒、0 の部分が白で印刷されます。

まず最初に、スタック経由で渡されている引数を受け取ることにします。ユーザープロセスの中ではレジスタを破壊してもかまわないので、引数は A3 と A4 に収めておくことにします。

userProc:

```

    move.l   4(sp),a4                *印刷環境レコードへのハンドル
    move.l   8(sp),a3                *描画範囲のレクタングルへのポイン
                                     タ

```

続いて、現在描画すべき範囲を知るためにカレントグラフポートを得ます。

```

    SXCALL   $A132                    * _GMGetGraph
    move.l   a0,a2                    * カレントグラフポートへのポインタ

```

グラフポートレコードの先頭のビットマップレコードへのポインタを得て、描画すべき範囲などの数値を計算します。

move.l	(a2),a0	* ビットマップへのポインタ
move.w	4(a0),d1	* Y 先頭
move.w	8(a0),d2	* Y 終端
move.w	6(a3),d0	* 描画範囲の Y 終端
cmp.w	d2,d0	* 描画範囲のほうが広い?
bge	userProc0	* ならば userProc0 へ
move.l	d0,d2	
userProc0:		
sub.w	d1,d2	
subq.w	# 1,d2	* D2: ライン数 -1
and.w	#\$ff,d1	* D1: このラインのビットパターン
moveq	# 0,d0	
move.w	6(a0),d0	* (X 終端
sub.w	2(a0),d0	* -X 先頭)
divu	# 8,d0	* ÷8
swap	d0	* 半端なドット数
move.b	#\$ff,d4	
lsl.w	d0,d4	
not.b	d4	* D4: 右端の 1 バイト用マスク
move.w	14(a0),d3	* D3: 1 ラインのバイト数
move.l	10(a0),a1	* A1: ビットマップのベースアドレス

以上で描画に必要な数値は得られたので、描画を行います。

userProc1:		
move.w	d3,d0	* 1 ラインのバイト数→カウンタ
subq	# 1,d0	* DBRA 用に-1
move.l	a1,a0	* ベースアドレス→ポインタ
userProc2:		
move.b	d1,(a0)+	* ビットマップに書き込む
dbra	d0,userProc2	* 1 ライン全部に書き込むまでループ
and.b	d4,-1(a0)	* 右端の不要な部分をマスク
lea	(a1,d3),a1	* 次の行へ


```

addq.w    #1,d1          *次の行のビットパターン
dbra      d2,userProc1   *すべてのラインに書き込むまで
                                ループ

```

描画が完了したら、DO に 0 を収めて RTS します。

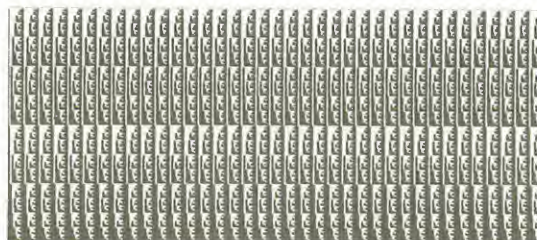
```

moveq     #0,d0
rts

```

参考までに、このようなユーザープロセスを登録した場合の印刷結果を図 13 に示しておきます。

■図 13 例のユーザープロセスによる印刷結果



テキストファイルの印刷

リソース PRTM を利用してテキストファイルを出力する方法については、前著『SX-WINDOW〜』222 ページで述べましたが、文章による解説だけでしたので、ここでコード例を示して補足しておきます。解説は省略して、コード例のみを示しておきますので、『SX-WINDOW〜』、あるいは注釈を参照してください。

TextPrint:

```

move.w    #-1,-(sp)      *すべてのタスクを検索
pea       prtManName(pc) *'prtman.r' というタスク名を
SXCALL    $A3F4          *__TSFindTskn
addq.l    #6,sp
tst.w     d0              *すでに存在するか?
bge       TextPrint0     * 存在するなら TextPrint0 へ
clr.w     -(sp)          *ID: 0

```

move.l	#'PRTM',-(sp)	* TYPE : PRTM
clr.l	-(sp)	* 環境
clr.l	-(sp)	* コマンドライン
pea	prtManName(pc)	* タスク名 : 'prtman.r'
move.w	#\$0001, -(sp)	* リソースから読み込み/起動
SXCALL	\$A351	* __TSFock
lea	20(sp), sp	
TextPrint0:		
move.w	d0, prtManID(a5)	* タスク ID を保存
pea	msgRec(a5)	* メッセージレコードを作成する場所
clr.w	-(sp)	* Hmodel : 0, Hmode2 : 0
move.w	#\$6e, -(sp)	* メッセージコード : \$6e
move.l	#4, -(sp)	* コマンドコード : 4
pea	fileName(pc)	* 印刷するファイル名 (ASCIIZ)
SXCALL	\$A361	* __TSMakeEvent
lea	16(sp), sp	
TextPrint1:		
move.w	#1, -(sp)	* 返事要
pea	msgRec(a5)	* メッセージレコード
move.w	prtManID(a5), -(sp)	* PRTM のタスク ID
SXCALL	\$A35F	* __TSCommunicate
addq.l	#8, sp	
cmp.w	#\$ffff, d0	* エラー ?
beq	TextPrint9	* ならば TextPrint9 へ
cmp.w	#\$fffe, d0	* 受け付けられなかった ?
beq	TextPrint1	* ならば TextPrint1 へ
cmp.w	#\$71, msgRec+14(a5)	* 返事のイベントコードは \$71 ?
bne	TextPrint9	* でなければ TextPrint9 へ
move.l	taskID(a5), d0	* 自分のタスク ID
cmp.w	msgRec+4(a5), d0	* 引数 1 は自分のタスク ID ?
bne	TextPrint9	* でなければ TextPrint9 へ
pea	msgRec(a5)	* メッセージレコードを作成する場所
clr.w	-(sp)	* Hmodel : 0, Hmode2 : 0
move.w	#\$6f, -(sp)	* メッセージコード : \$6f
clr.l	-(sp)	* 引数 2 なし

```

        clr.l      -(sp)                * 引数 1 なし
        SXCALL    $A361                * __TSMakeEvent
        lea       16(sp),sp

TextPrint3:
        move.w    #0,-(sp)            * 返事不要
        pea      msgRec(a5)           * メッセージレコード
        move.w    prtManID(a5),-(sp)   * PRTM のタスク ID
        SXCALL    $A35F                * __TSCommunicate
        addq.l    #8,sp

        cmp.w     #$fffe,d0           * 受け付けられなかった?
        beq       TextPrint3          *   ならば TextPrint3 へ

TextPrint9:

        :

prtManName:
        dc.b      'prtman.r',0

```

ハードコピー

リソース PRTM の ID0 を prtman.r の名前で起動することによって、デスクトップ全体をハードコピーすることができます。このとき、コマンドラインとして -H を指定する必要があります。

このときのコードは次のようになります。

```

HardCopy:
        move.w    #0,-(sp)            * リソース ID : 0
        move.l    #'PRTM',-(sp)       * リソースタイプ : PRTM
        clr.l     -(sp)                * 環境へのポインタ
        pea      HCOption(pc)         * コマンドラインへのポインタ
        pea      prtManName(pc)       * ファイルネームへのポインタ
        move.w    #$00_01,-(sp)       * 起動モード : リソースから起動
        SXCALL    $A351                * __TSFock
        lea       20(sp),sp

        :

HCOption:
        dc.b      2,'-H'

prtManName:
        dc.b      'prtman.r',0

```

4 プリントドライバの作成

新しくプリントドライバを作成するというのはまれであると思われるので、プリントドライバが満たすべき仕様を示すのみにとどめておきます。

ドライバの形式

ドライバはリロケータブルでリエントラントにつくられていなければなりません。SX1.02の名残りで R 型のプログラムモジュールに似たヘッダを先頭に置く必要がありますが、モジュールとして起動されることはありません。

ヘッダの形式は以下のとおりです。

オフセット	欄 名	内 容
+\$00.l	pType	固定文字列 'OBJR'
+\$04.l	pcSize	プログラムエリアのサイズ
+\$08.l	pExec	スタートアドレスオフセット (ダミー)
+\$0c.l	pdSize	ワークエリアのサイズ
+\$10	pRsv	システム予約 (3 ロングワード)
+\$1c.l	prtDrvName	プリンタ名 (ASCII) へのオフセット
+\$20.l	prtDrvStart	ドライバスタートアドレスオフセット
+\$24.l	prtDrvType	固定文字列 PRTD
+\$28.w	prtDrvVer	ドライバのバージョン
+\$2a.w	prtDrvExt	システム予約

ドライバのコントロール

プリントマネージャがドライバを利用するときには、ドライバスタートオフセットで示されたエントリを呼び出します。その際、AO にはパラメータ領域へのポインタが収められています。

パラメータ領域の形式は以下のとおりです。

オフセット	欄 名	内 容
+\$00.w	command	ドライバコマンド
+\$02.l	p1	パラメータ 1 } コマンドによって意味が異なる パラメータ 2 } パラメータ 3 } パラメータ 4 }
+\$06.l	p2	
+\$0a.l	p3	
+\$0e.l	p4	

ドライバコマンド

ドライバコマンドは、次の 17 個が存在します。

●command=0: PZ_NIT

パラメータ

なし

返り値

DO.L =0 正常終了

=-1 異常終了

プリンタドライバを初期化します。

●command=1: PZ_TINI

パラメータ

なし

返り値

DO.L =0 正常終了

=-1 異常終了

プリンタドライバの終了処理を行います。

●command=2: PZ_CTRL

パラメータ

p1 サブコマンド

p2 パラメータ 1

p3 パラメータ 2

p4 パラメータ 3

返り値

DO.L =0 正常終了

=-1 異常終了

サブコマンドによって指定したプリンタの直接制御を行います。

・サブコマンド=0: PD_RESET

プリンタを初期化します。

・サブコマンド=1: PD_CRLF

改行します。パラメータ 1 には、1/120 インチ単位で改行幅が入ります。パラメータ 1 が -1 の場合は 1/6 インチ改行を行います。

・サブコマンド=2: PD_FF

改ページします。

・サブコマンド=3: PD_THRU

パラメータ 1 には印刷環境レコードへのハンドル、パラメータ 2 にはデータへのハンドル、パラメータ 3 にはデータのバイト数が入ります。パラメータで指定したデータを、そ

のままプリンタに出力します。

●command=3 : PZ_DEFAULT

パラメータ

p1 印刷環境レコードへのハンドル

返り値

DO.L =0 正常終了

=-1 異常終了

AO.L 印刷環境レコードへのハンドル

印刷環境レコードにデフォルトの情報をセットします。

●command=4 : PZ_VALIDATE

パラメータ

p1 印刷環境レコードへのハンドル

返り値

DO.L =1 正常終了：変更した

=0 正常終了：変更せず

=-1 異常終了

AO.L 印刷環境レコードへのハンドル

印刷環境レコード内の情報が正しいかどうかチェックし、正しくない場合は調整します。

201 ページ図 14 のフローチャートのような流れでチェック/調整を行います。

●command=5 : PZ_IMGDLOG

パラメータ

p1 印刷環境レコードへのハンドル

返り値

DO.L =1 正常終了：変更した

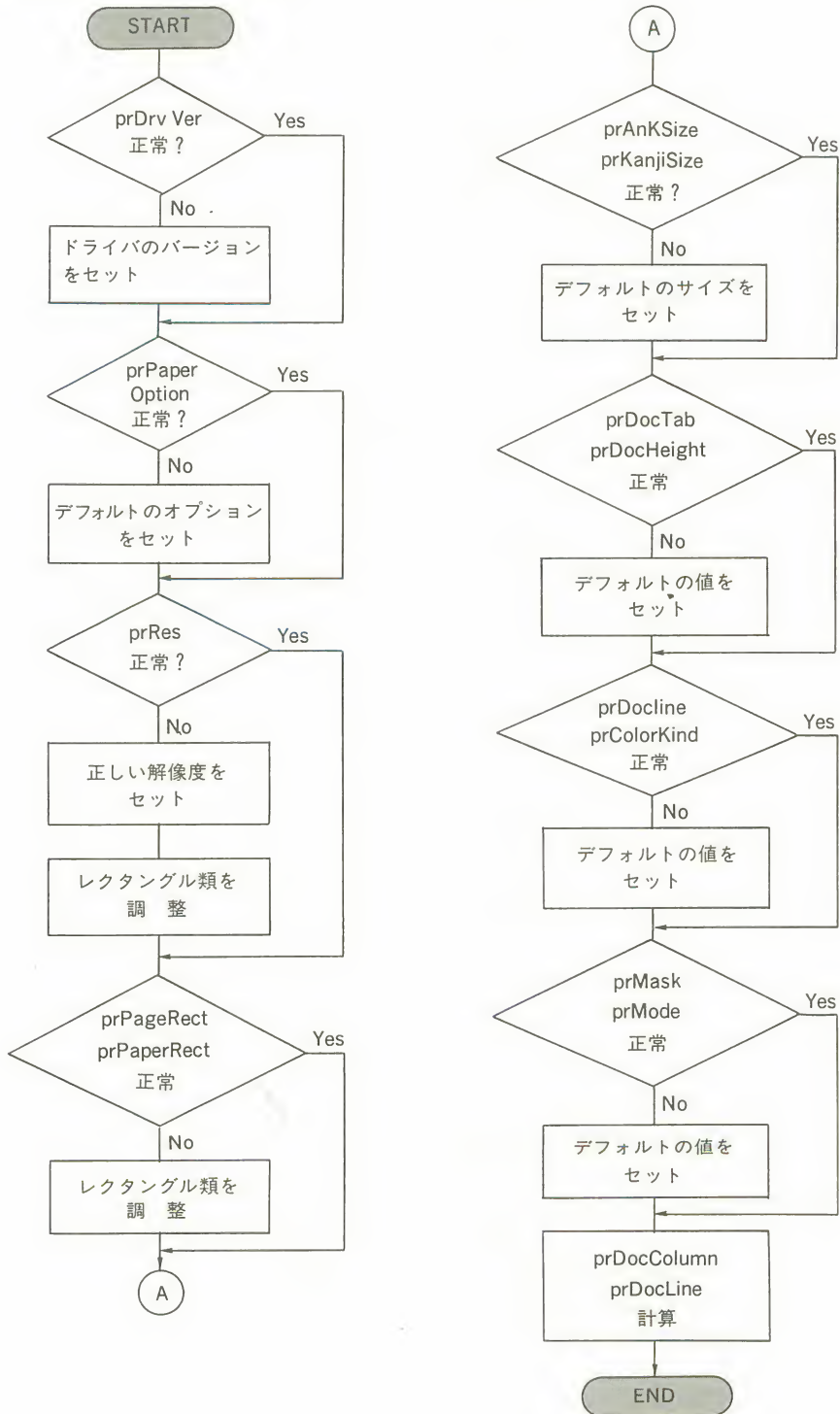
=0 正常終了：変更せず

=-1 異常終了

AO.L 印刷環境レコードへのハンドル

ページ印刷用の印刷環境設定ダイアログをオープンし、マウスによる操作にしたがって印刷環境レコードの内容を変更します。

■図 14 PZ_VALIDATE の処理の流れ



●command=6 : PZ_STADLOG

パラメータ

p1 印刷環境レコードへのハンドル

返回值

DO.L =1 正常終了：変更した

 =0 正常終了：変更せず

 =-1 異常終了

AO.L 印刷環境レコードへのハンドル

ページ印刷用の印刷環境設定ダイアログをオープンし、マウスによる操作にしたがって印刷環境レコードの内容を変更します。

●command=7

未定義です。

●command=8 : PZ_OPENIMG

パラメータ

p1 印刷環境レコードへのハンドル

返回值

DO.L =0 正常終了

 =-1 異常終了

AO.L グラフポートへのポインタ

ページ印刷用のグラフポートを作成します。

●command=9 : PZ_RECORDPG

パラメータ

p1 レクタングルレコードへのポインタ

返回值

DO.L =0 正常終了

 =-1 異常終了

ページ印刷用のスクリプトの記録を開始します。

p1 に渡されたレクタングルを、ビットマップレクタングルとするビットマップレコードを作成し、グラフポートにセットします(\$A1C8 GMCalcBitmap, \$A1D1 GMCalcGraph を使用)。p1 が 0 の場合、印刷環境レコード中の prPageRect をビットマップレクタングルとします。以上の処理の後、\$A199 GMOpenScript を呼びます。

●command=10 : PZ_PRINTPG

パラメータ

p1	=0	スクリプト記録終了
	≠0	スクリプト廃棄
p2		ユーザープロセスへのポインタ返り値
DO.L	=0	正常終了
	=-1	異常終了

スクリプトの記録を終了し、印刷を開始します。

p1 が 0 以外の場合は、スクリプトを廃棄し、印刷は行いません。

p2 が 0 の場合、スクリプトからビットイメージを展開します。0 以外の場合、ユーザープロセスへのポインタであると解釈し、印刷範囲を示すレクタングルレコードへのポインタと印刷環境レコードへのハンドルをスタックに積んで、そのアドレスを呼び出します。どちらの場合も、その後、プリンタへの出力を行います。

●command=11 : PZ_ACTION

パラメータ

p1	サブコマンド
----	--------

返り値

DO.L	=0	印刷終了
	=1	印刷中
	=2	印刷中断
	=3	タイムアウト発生
	=-1	異常終了

サブコマンドによって指定した印刷の制御を行います。

・サブコマンド=0 : PC_STAT

印刷を続行します。

スクリプトを展開、あるいはユーザープロセスを呼び出しつつ、データを出力します。

・サブコマンド=1 : PC_END

印刷を終了します。

・サブコマンド=2 : PC_STOP

印刷を中断します。

・サブコマンド=3 : PC_CONT

印刷を再開します。

●command=12 : PZ_CLOSEIMG

パラメータ

なし

返回值

DO.L =0 正常終了

=-1 異常終了

ページ印刷を終了します。

●command=13 : PZ_STRING

パラメータ

p1 印刷環境レコードへのハンドル

p2 文字列へのハンドル

p3 文字列のバイト数

p4 =0 文字列出力後に改ページする

=1 文字列出力後に改ページしない

返回值

DO.L =0 正常終了

=-1 異常終了

コード印刷を開始します。

●command=14 : PZ_VERSION

パラメータ

なし

返回值

DO.L バージョン (下位ワード)

プリンタドライバのバージョンを返します。

●command=15 : PZ_MAXRECT

パラメータ

p1 印刷環境レコードへのハンドル

p2 用紙の種類 (176 ページ参照)

p3 レクタングルレコードへのポインタ

返回值

DO.L =0 正常終了

=-1 異常終了

p2 で指定された用紙に印刷可能な最大の範囲を、p3 で指定されたレクタングルレコード

に収めます。

● command=16 : PZ_STATUS

パラメータ

なし

返り値

DO.1 =0 出力不可

 =1 出力可

プリンタに出力可能かどうかを返します。

5

まとめ

プリンタは日に日に高性能、低価格になり、いまや 300DPI 以上の解像度のプリンタを非常に安価に手に入れることができるようになりました。プリントマネージャでは、こうした高性能のプリンタを生かして、高品質の出力を行うための工夫がなされていることがわかりいただけたと思います。

プリントマネージャによって、SX-WINDOW 上の DTP 環境の基盤が準備されたといえるかもしれません。

3²

サブウィンドウマネージャ

1つのアプリケーションが複数のウィンドウを開いて使用することは以前にもありました。しかし、ピンボール、Xのように、たかだか2つのウィンドウ程度ならともかく、複雑なアプリケーションになると、より多くのウィンドウを開かなければならないことも考えられます。従来の形式のウィンドウを複数並べた場合、どのウィンドウが作業の中心になるのかわかりにくくなる場合があります。

サブウィンドウマネージャは、「サブウィンドウ」という概念を導入することにより、こうした混乱に1つの解決策を提示します。

1

サブウィンドウの意味

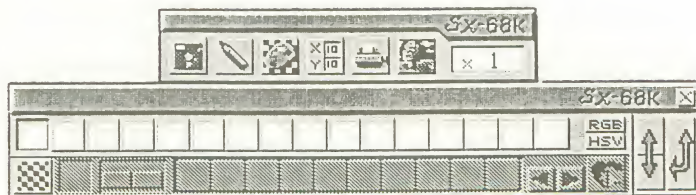
サブウィンドウがどのようなものか理解するには、Easypaintを使ってみるのがいちばんの早道でしょう。Easypaintを立ち上げると、作画ウィンドウと呼ばれる標準ウィンドウのほかに、ツール類を収めたいくつかの見慣れない形のウィンドウが現れます。これらは、サブウィンドウによって実現されています。これらを例にとって、サブウィンドウの特徴を述べることにします。

これらのウィンドウ（本来のウィンドウと区別するために、以降、サブウィンドウと呼びます）は、見かけも、ドラッグなどの操作感覚も、ほとんどウィンドウと同じですが、しばらく使っていると、いくつか違いが見えてきます。

まず最初に気がつくことは、ウィンドウの枠が見慣れない形をしていることです。これは標準ウィンドウやプレーンウィンドウなど、ウィンドウ定義関数によって用意されているウィンドウのどれとも異なっています。SX1.10になって、このような形式のウィンドウが追加されたわけでもありません。つまり、ウィンドウ定義関数とは関係のない、自由な形のウィンドウであるといえます（図1）。

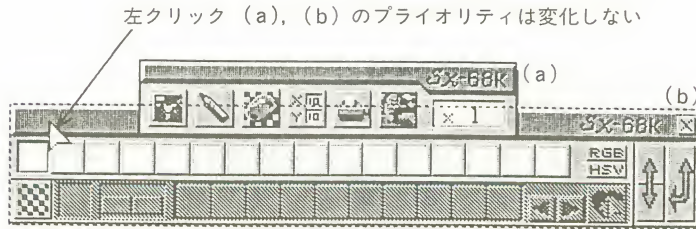
サブウィンドウは、表示されているときにはつねに作画ウィンドウよりも手前に表示されています。Easypaintでは、複数のサブウィンドウがデスクトップ上に置かれる場合があります。

■図1 サブウィンドウは枠の形が自由



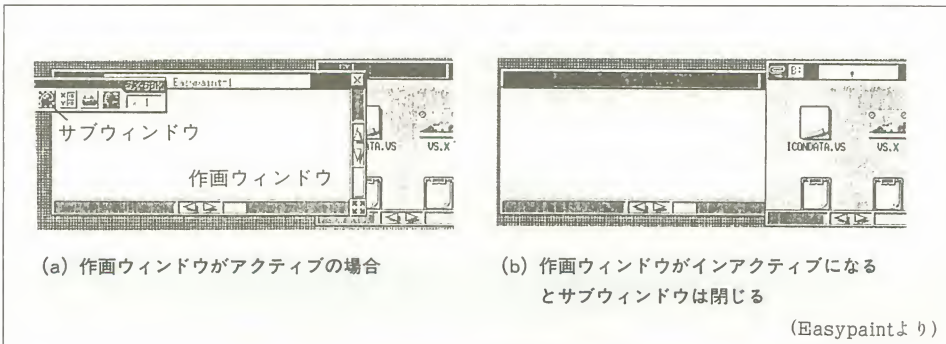
すが³、それらにはアクティブ/インアクティブの区別がなく、その前後関係が変化することはありません (図 2)。

■図 2 アクティブ/インアクティブの概念がない



さらに、作画ウィンドウがインアクティブになった場合、サブウィンドウはデスクトップから一時的に取り除かれ、ふたたび作画ウィンドウがアクティブになるまで表示されません (図 3)。

■図 3 ウィンドウのアクティベートによって閉じる



以上は、いずれもサブウィンドウの特徴がよく表れている箇所です。

サブウィンドウの目的は、おもな作業場となるウィンドウ (主ウィンドウ) とは別のウィンドウを用意し、その中にアイコンや各種情報の表示領域を用意することによって、わかりやすいユーザーインタフェースを構築することにあります。ウィンドウの一種ですから、ユーザがもっとも操作/参照しやすい場所にドラッグしておくことも可能で、その自由度はかなり高いものとなります。

先ほど挙げた特徴は、サブウィンドウがそういった用途のために用意されたことを物語っています。

(1) ウィンドウの枠の形を自由に設定できる

標準ウィンドウを利用して複数のウィンドウを操る場合、デスクトップ上に標準ウィンドウがいくつも存在することになり、どれが主ウィンドウであるのか区別が付きにくくなります。

サブウィンドウならば、ウィンドウの枠を自由な形にすることができるので、主ウィンドウと一目で区別できるような形を与え、混乱を防ぐことができます*1。

*1:逆に自由すぎることから混乱を招く可能性もあります。せめて同じアプリケーションで扱うサブウィンドウには統一性を持たせたほうがよいでしょう。

(2) アクティブ/インアクティブの概念がない

アイコンを収めたウィンドウ内部、または主ウィンドウをクリックしたとき、アクティベートが発生して前後関係が入れ替わったりすると、その位置関係によっては画面が見づらくなったり、操作が煩雑になったりすることがあります*2。サブウィンドウにはアクティブ/インアクティブの概念がないので、つねに主ウィンドウ、そしてもちろん、ほかのアプリケーションのウィンドウよりも手前に表示されます。

*2:わかりやすい例として、主ウィンドウがアクティブになった結果、ツール類を収めたウィンドウがその後ろに隠れてしまうことが挙げられます。

(3) 主ウィンドウのアクティベートによって閉じる

いままで述べてきたような用途のウィンドウは、主ウィンドウがアクティブの場合には必要ですが、インアクティブになった場合には必要がないばかりか、ほかのアプリケーションの操作の邪魔になります。したがって、主ウィンドウがインアクティブになった場合は一時的に閉じられたほうが好都合なのです。

このような特徴から、主ウィンドウに従属するように見えるウィンドウ、サブ（副）ウィンドウという名称に納得していただけたらと思います。

2

サブウィンドウの仕組み

まず最初に、サブウィンドウに関する情報をまとめた、サブウィンドウレコードの内容を示しておくことにします。ウィンドウ 1 枚 1 枚について、それぞれウィンドウレコードが存在していたように、サブウィンドウをオープンすることで、それぞれにサブウィンドウレコードが用意されます。

サブウィンドウレコードの内容は以下のとおりです。

オフセット	欄 名	内 容
+\$00	pix	グラフポートレコード
+\$40.w	wKind	ウィンドウの種類 (\$20 固定)
+\$42.b	wVisible	描画状態
+\$43.b	wHilite	<使用されない>
+\$44.b	wClose	<使用されない>
+\$45.b	wStatus	<使用されない>
+\$46.w	wOption	<使用されない>
+\$48.l	wOutSide	アウトサイドリージョンへのハンドル
+\$4c.l	wInside	インサイドリージョンへのハンドル (デフォルトでは wOutSide=wInside)
+\$50.l	wUpdate	アップデートリージョンへのハンドル
+\$54.l	wDef	<使用されない>
+\$58.l	wDefData	<使用されない>
+\$5c.l	wTitle	<使用されない>
+\$60.w	wTWidth	<使用されない>
+\$62.l	wControl	コントロールへのハンドル
+\$66.l	wNext	次のサブウィンドウへのポインタ
+\$6a.l	wPicture	ウィンドウスクリプトへのハンドル
+\$6e.l	wTask	<使用されない>
+\$72.l	tPrio	プライオリティ値

ウィンドウレコード
と同じ形式

一見してわかるように、サブウィンドウレコードはウィンドウレコードとほとんど同じ形式です。異なる点はウィンドウ定義関数に関わる部分で、これらの部分は使用されていません。また、プライオリティ値という新しい欄が追加されています。

サブウィンドウは、デスクトップ上においてアプリケーションが自由に使うことのできる領域という意味で、ウィンドウの一種であるといえます。実際、サブウィンドウは、ほとんどウィンドウと同じ仕組みのもとで動作していますが、先ほど挙げた 3 つの特徴が両者を異なるものとしています。この 3 つの特徴の仕組みを説明することでサブウィンドウのメカニズムを明らかにすることにしましょう。

(1) ウィンドウの枠の形を自由に設定できる

＝ウィンドウ定義関数を利用しない

ウィンドウでは、ウィンドウ ID を指定することによって、何種類か用意されているウィンドウのうち、目的に応じたものを開きます。しかし、サブウィンドウにはウィンドウの種類というものが存在しません*3。

*3: サブウィンドウレコードの中のウィンドウレコードに相当する部分に「ウィンドウの種類」が記録されていますが、ここには \$20 が収められ、サブウィンドウであることを示します。この値は固定で、変化することはありません。

ウィンドウを開くと、ウィンドウの種類に応じて四角形のウィンドウの枠がデスクトップ上に描かれ、その内部のウィンドウコンテンツがアプリケーションのための領域として提供されました。しかし、サブウィンドウではウィンドウの枠は描画されません。さらに、サブウィン

ドウの内部が背景色で塗り潰されることもありません。

これらはすべて、ウィンドウ定義関数とサブウィンドウの間になんの関係も存在しないことを意味しています。したがって、サブウィンドウの枠の描画や、サブウィンドウのドラッグなどの処理は、アプリケーションが自分で行わなければなりません。サブウィンドウに関する情報はサブウィンドウレコードに含まれます。このレコードはウィンドウレコードとはほぼ同じ形式であるため、やはり先頭\$40バイトにグラフポートを含みます。したがって、サブウィンドウ内部にはグラフィックマネージャを利用して自由に描画することが可能です。サブウィンドウの枠の描画などは、グラフィックマネージャを利用して描画することになります(図4)。

■図4 サブウィンドウの枠はアプリケーションが描画する



これらはすべて Easypaint 自身が描画している (Easypaint より)

(2) アクティブ/インアクティブの概念がない

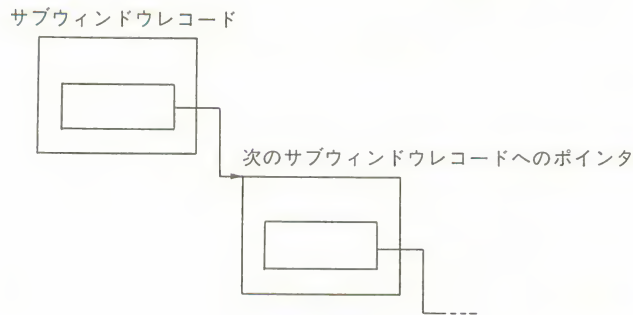
=サブウィンドウリスト、プライオリティの導入

ウィンドウのアクティブ/インアクティブの概念は、ウィンドウマネージャのウィンドウリストに強く依存しています。ウィンドウリストはデスクトップ上に存在するウィンドウの前後関係を示すデータ構造で、アクティベートとは、すなわち、このリストに変化が起こったことを意味します。個々のウィンドウから見れば、アクティベートの際、自分がウィンドウリストの一方の端、つまり画面のもっとも手前に来ればアクティブということになります。逆に、画面のもっとも手前でなくなればインアクティブです。

サブウィンドウは、ウィンドウマネージャのウィンドウリストとはまったく別なデータ構造、サブウィンドウリストによって連結されています。構造的にはウィンドウマネージャと同様で、211 ページの図5のように表現することができます。

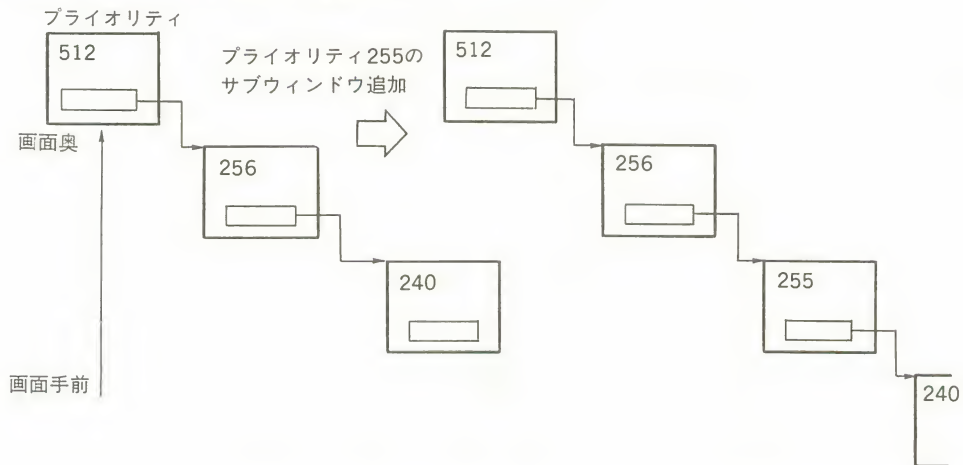
ウィンドウリストがウィンドウの前後関係を意味していたように、サブウィンドウリストもサブウィンドウどうしの前後関係を意味しています。図5の表現では、下のほうに位置するサブウィンドウがもっとも手前に表示されることになります。ウィンドウリストは、ウィンドウがいくつも開かれて、その前後関係が変化する中で形作られますが、サブウィンドウリストでは、あらかじめ、ある程度その要素(サブウィンドウ)の順番が決まっており、その前後が入れ替わるということはありません。その順番、すなわち前後関係を決めるのがプライオリティです。

■図5 サブウィンドウリストの概念



新しくサブウィンドウを開くときには、同時にそのサブウィンドウのプライオリティを指定します。プライオリティは、0～\$FFFFFFFF の符号なしのロングワード値で、小さいほど手前に表示されることになります。新しく開かれたサブウィンドウのサブウィンドウレコードは、プライオリティにしたがってサブウィンドウリストのしかるべき位置に挿入されます(図6)。

■図6 サブウィンドウリストへの登録



このとき、もしもプライオリティが同じ値のサブウィンドウがすでに登録されていた場合でも、問題なく登録されますが、後から登録したサブウィンドウのほうが手前に表示されることになります。

プライオリティによって決められたサブウィンドウの前後関係は、サブウィンドウリストに連なっているかぎり変化することはありません*4。

*4: サブウィンドウの前後関係を変化させたい場合は、一度サブウィンドウリストから外して、サブウィンドウレコード中のプライオリティ値を変化させた後、ふたたびサブウィンドウリストに登録するといった処理を行います。

(3) 主ウィンドウのアクティベートによって閉じる

=ウィンドウマネージャとの関係

主ウィンドウがインアクティブになった場合に、それに従属するサブウィンドウが閉じる仕組みについて解説します。

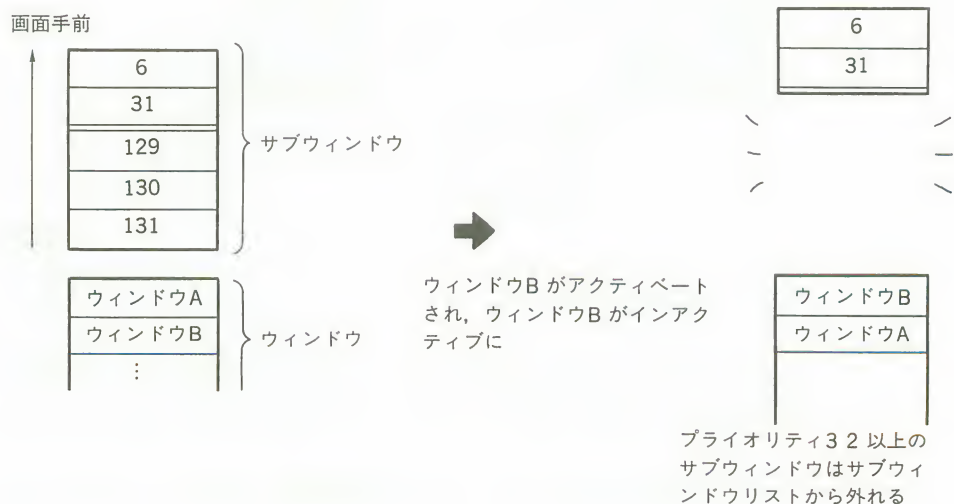
先ほどウィンドウリストとサブウィンドウリストはまったく別であると述べましたが、いっさい関係がないわけでもありません。サブウィンドウリストは、ウィンドウリストの「手前」側の端に接ぎ木されていると考えてもよいでしょう。このため、つねにサブウィンドウはアクティブなウィンドウよりも手前に表示されます。

アクティベートが発生し、アクティブだったウィンドウがインアクティブに変化すると、サブウィンドウはサブウィンドウリストから自動的に外されます*5。このとき、例外があつて、プライオリティが32未満のサブウィンドウはリストに残ります*6 (図7)。サブウィンドウリストから外されたサブウィンドウは、デスクトップから除かれ、ユーザの目からは消えてしまったように見えます。もちろん、このときプライオリティが32未満のサブウィンドウは消えずに残っています。

*5:すべてのサブウィンドウを消したくないが、ウィンドウのアクティベートを行いたいときのために、ウィンドウマネージャに \$AIFW WMSelect2 という SX コールが増設されています。

*6:プライオリティ 0~15 はシステム予約です。したがって、消えないサブウィンドウをアプリケーションが使用したい場合はプライオリティ 16~31 を使用することになります。

■図7 アクティベートによるサブウィンドウリストの変化



消える場合は自動ですから問題はありませんが、主ウィンドウがアクティブになったときにサブウィンドウを再表示させたい場合は、アプリケーションが判断し、再表示したいサブウィンドウをサブウィンドウリストに登録する処理を行う必要があります。

このような仕組みによって、複数のアプリケーションがサブウィンドウを利用する場合でも問題が発生しないことを確認しておきます。

すでに2つのサブウィンドウ（プラオリティ 255 と 256）を開いているアクティブなアプリケーション A が動作している状態で、あらたに1つのサブウィンドウ（プライオリティ 512）を持つアプリケーション B を起動し、その後、アプリケーション A がアクティベートされた場合を例にとって考えてみましょう。

(a) 初期状態

デスクトップにはアプリケーション A の主ウィンドウ、ウィンドウ A とサブウィンドウ 255 と 256 が存在しています（214 ページの図 8 (a) 参照）。

(b) アプリケーション B が起動され、ウィンドウ B が開かれる

これによって、ウィンドウ A はインアクティブとなり、ウィンドウ A を主ウィンドウとするサブウィンドウ 255 と 256 はサブウィンドウリストから取り除かれます。この結果、デスクトップにはインアクティブなウィンドウ A と、アクティブなウィンドウ B だけが残ります（図 8 (b)）。

(c) アプリケーション B がサブウィンドウ 512 を開く

サブウィンドウ 512 は、サブウィンドウリストに登録され、デスクトップ上に表れます（図 8 (c)）。

(d) アプリケーション A がアクティベートされる

ウィンドウ B はインアクティブとなり、サブウィンドウ 512 はサブウィンドウリストから外れ、デスクトップから消えます。

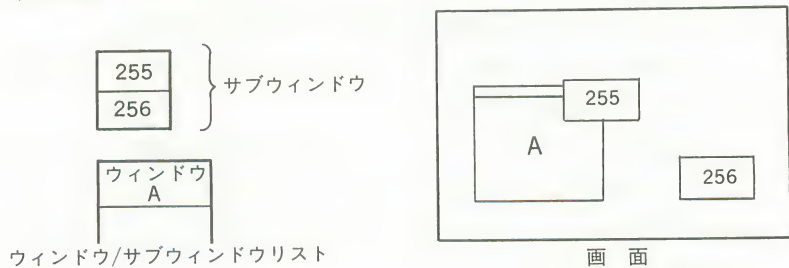
アクティベートイベントがアプリケーション A に通知され、ウィンドウ A がアクティブになったことを知ると、アプリケーション A はサブウィンドウ 255 と 256 をサブウィンドウリストに登録する処理を行います。この結果、デスクトップにはアクティブなウィンドウ A、それに従属するサブウィンドウ 255、256、そしてインアクティブなウィンドウ B が表示されていることとなります（214 ページ図 8 (d)）。

さらにウィンドウ B がアクティブになった場合、どのような処理が行われ、デスクトップがどのような様子になるのか、説明するまでもないと思います。

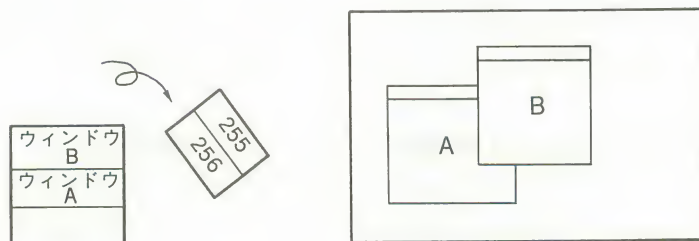
このように、うまくサブウィンドウの出し入れを行うためには、アプリケーションが次の2つの点を守っていることが条件となります。

■図8 複数のアプリケーションとサブウィンドウ

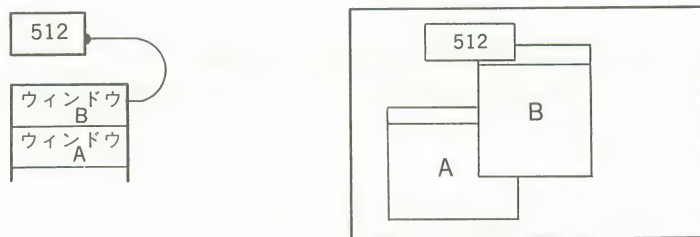
(a) 初期状態



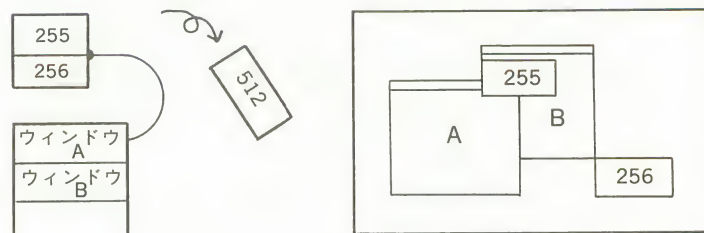
(b) アプリケーションB が起動され、ウィンドウB が開かれる



(c) アプリケーションB がサブウィンドウ512を開く



(d) アプリケーションA がアクティベートされる



(1) 主ウィンドウとなるべきウィンドウがアクティブな状態でサブウィンドウを開く

サブウィンドウや主ウィンドウにその主従関係が記録されているわけではなく、サブウィンドウマネージャとウィンドウマネージャの連係がそのような関係をつくり出していることはすでにおわかりいただけたと思います。連係の内容を考えれば、この条件の意味もおのずから明らかでしょう。

(2) アクティベートイベントで適切な処理を行う

主ウィンドウがアクティブになった場合、それに従属するサブウィンドウをサブウィンドウリストに登録する処理を行います。主ウィンドウがインアクティブになったときにサブウィンドウリストから外す処理は、サブウィンドウマネージャが自動的に行います。

3**サブウィンドウの利用**

プログラミング的には、サブウィンドウはウィンドウとほぼ同様に扱うことができます。ただし、いくつか注意が必要な箇所がありますので、ここで解説しておきます。

ほかのマネージャとの相性

何度も述べてきたように、サブウィンドウとウィンドウは非常に似た存在です。サブウィンドウレコードとウィンドウレコードにはかなりの互換性がありますが、その相違のためにウィンドウレコードを引数として指定するマネージャとの相性は、次のようになっています。

○グラフィックマネージャ

サブウィンドウレコードの先頭のグラフポートレコードを通して、サブウィンドウ内のビットマップに描画することができます。すべての SX コールが使用可能です。

△ウィンドウマネージャ

ウィンドウ定義関数を利用できないことから、ウィンドウマネージャの SX コールの大部分は利用不可です。

例外的に、\$A20D WMUpdate、\$A20E WMUpdtOver などのアップデートリジョン関係、ウィンドウ定義関数と無関係な、ユーティリティ的な SX コール (\$A225 WMDragRgn など) は利用可能です。

○コントロールマネージャ

すべての SX コールが使用可能です。

○テキストマネージャ

すべての SX コールが使用可能です。

△タスクマネージャ

タスクマネージャのユーティリティの中に、ウィンドウレコードを引数とするものがあります。おもなものについて使用の可否を示します。

×\$A3A2 SXCallWindM
 ○\$A3A3 SXCallCtrlM
 ×\$A3AA SXInvalScBar
 ×\$A3AB SXValidScBar
 ×\$A41F SXCallWindM2

サブウィンドウの利用

サブウィンドウはウィンドウと同じように利用できます。

●サブウィンドウのオープン

サブウィンドウをオープンする際には、すでに主ウィンドウがオープンされ、アクティブになっている必要があります*7。

*7: プライオリティ 0~31 の消えないサブウィンドウを使う場合、主ウィンドウに従属するものとしての使い方以外も考えられます。

サブウィンドウのオープンには、ウィンドウのオープンに\$A1F9 WMOpen を使うように、\$A227 WSOpen を使います。WSOpen の引数は次のとおりです。

long	sWinPtr	*サブウィンドウレコードのアドレス
long	rgnHdl	*サブウィンドウの内部となるリージョンへのハンドル
long	priority	*プライオリティ値

sWinPtr は、\$A1F9 WMOpen と同様に、0 を指定するとヒープ領域に、アドレスを指定すると、そのアドレスからサブウィンドウレコードを作成します。

ウィンドウの外形は、レクタングルレコードではなく、リージョンで指定します。ということは、リージョンで表現できる領域であれば、どんな形でもサブウィンドウの形にすることが可能となっています。グローバル座標系です。

プライオリティ値は 0~\$FFFFFFF ですが、0~16 はシステム予約です。

実際にコードを書くと、次のようになります。ここではごくオーソドックスに、四角形のサブウィンドウを作成してみましょう。

まず、リージョンを作成します。

SXCALL	\$A15A	* __GMNewRgn
pea	rectPtr(pc)	* サブウィンドウの内部を示すレクタングルレコードへのポインタ
pea	(a0)	* 新しく作成したリージョンへのハンドル
SXCALL	\$A15F	* __GMRectRgn
addq.l	# 8,sp	
move.l	a0,rgnHdl(a5)	* リージョンへのハンドルを保存

作成した、四角形の領域を表現するリージョンを使って、サブウィンドウをオープンします。プライオリティは 255 とし、サブウィンドウレコードはヒープゾーンに作成することになります。

move.l	# 255,-(sp)	* プライオリティ値
move.l	rgnHdl(a5),-(sp)	* 内部を意味するリージョンへのハンドル
clr.l	-(sp)	* ヒープゾーンに作成
SXCALL	\$A227	* __WSOpen
lea	12(sp),sp	

この結果、正常にオープンできた場合は AO にサブウィンドウレコードへのポインタが返ってくるので、ワークに保存しておきます。

```
move.l    a0,sWinPtr(a5)
```

● サブウィンドウ内部の描画

サブウィンドウ内部へ描画するためのグラフポートは、サブウィンドウレコードの先頭に埋め込まれています。したがって、サブウィンドウレコードの先頭アドレスをグラフポートレコードのアドレスとしてセットすることにより、以降、内部への描画を自由に行うことができます。

move.l	sWinPtr(a5),-(sp)	
SXCALL	\$A131	* __GMSetGraph
addq.l	# 4,sp	

ここで忘れてはならないのが、サブウィンドウの枠などはいっさい描画されていないことです。必要があれば、自分でサブウィンドウの枠や付属物を描画し、内部を背景色で塗り潰さなければなりません。

●アップデートイベントへの対応

サブウィンドウ内部にアップデートが必要になった場合は、ウィンドウ同様にアップデートイベントが発生します。アップデートの方法はウィンドウと同様ですが、サブウィンドウの枠なども自分で書き直す必要があることに注意してください。

サブウィンドウの枠や内部の描画が、DrawSub というサブルーチンで一括して行われている場合、アップデート処理は次のように書くことができます。

```

move.l    sWinPtr(a5),-(sp)      * アップデート開始
SXCALL    $A20D                  * __WMUpdate
addq.l    # 4,sp

brsr      DrawSub                * 枠 & 内部を書き直すサブルーチン

move.l    sWinPtr(a5),-(sp)      * アップデート終了
SXCALL    $A20E                  * __WMUpdtOver
addq.l    # 4,sp

```

●アクティベートイベントへの対応

プライオリティ値 32 以上の「消える」サブウィンドウは、主ウィンドウがアクティブになった際には、サブウィンドウリストへの再登録が必要です。サブウィンドウをサブウィンドウリストに登録するには \$A22A WSEnlist を使用します。このコールで登録できるのは、サブウィンドウリストから外れているサブウィンドウだけなので注意してください。

本書で示すスケルトンのアクティベートイベント処理ルーチンでは、主ウィンドウのアクティブフラグが真のときに主ウィンドウがアクティブになった場合*8 に不都合が生じるので、次のように書き換えてしまえばよいでしょう。

*8: ウィンドウをもっとも手前にオープンすると、その直後にアクティベートイベントが発生します。スケルトンではアクティブフラグを真で初期化しているので、こういうケースが発生してしまうことになります。

```

ACTIVATE:                                * [アクティベートイベント]

move.l    eventRec_whoml(a5),d0
beq       ACT9
lea       winPtr(a5),a0                  * 自分のウィンドウが
cmp.l     a0,d0                          * アクティブになった?
bne       ACT0                           * 違うのなら ACT0 へ
tst.b     winActive(a5)                  * すでにアクティブ?
bne       ACT9                           *   ならば ACT9 へ

st        winActive(a5)                  * アクティブフラグをセット

```



```

        move.l    sWinPtr(a5),-(sp)          *サブウィンドウを再登録
        SXCALL   $A22A                      * __WSEnlist
        addq.l    # 4,sp

        bra      ACT9

ACT0:
        sf       winActive(a5)              * アクティブフラグをリセット

ACT9:
        moveq     # 0,d0
        rts

```

● レフトダウンイベントへの対応

サブウィンドウのウィンドウとしての特性を生かすためには、ドラッグしたりできなければいけません。しかし、ウィンドウマネージャが利用できない以上、\$A205 WMDrag などを利用することはできません。したがって、そういった処理はすべて自分で行わなければなりません。

ここではコードは示しませんが、本章末のサンプルプログラムでドラッグの処理を行っていますので、参考にしてみてください。

サブウィンドウの内部にコントロールを置いたりした場合は、ウィンドウ同様\$A3A3 SXCallCtrlM が利用できるので、処理は非常に簡潔に記述できます。

● サブウィンドウのクローズ

サブウィンドウをクローズする場合、サブウィンドウは必ずサブウィンドウリストに登録されてなければいけないことに注意してください。また、オープン時にサブウィンドウレコードをヒープ上に作成したか、指定した場所に作成したかによって、使用する SX コールが異なります。ヒープ上に作成していた場合は\$A229 WSDispose、それ以外の場合は\$A228 WSClose です。

例の場合は、ヒープ上に作成していましたから、前者を使います。

```

        move.l    sWinPtr(a5),-(sp)
        SXCALL   $A229                      * __WSDispose
        addq.l    # 4,sp

```

サブウィンドウの内部を示していたリージョンは、サブウィンドウをオープンした時点で別のリージョンにコピーされて不要になっているのですが、まだ廃棄していなかった場合は、ここで廃棄しておいてください。

```
move.l    rgnHdl(a5),-(sp)
SXCALL    $A15B                      * __GMDisposeRgn
addq.l    #4,sp
```

4

まとめ

サブウィンドウは、自由度が高く、それだけに危険性をはらんだ存在でもあります。しかし、ユーザーインタフェースを向上させるという目的を見失わず、常識的な範囲内で、その自由度を生かすことを心がける必要があるでしょう。

3³ サンプルプログラム

プリントマネージャとサブウィンドウマネージャの利用例として、サンプルプログラム PRNSMPL と SWINSMPL を示します。

1 プリントマネージャのサンプルプログラム

プリントマネージャのサンプルプログラム PRNSMPL は、プロセス印刷を利用して特定のウィンドウの内容をハードコピーするプログラムです。

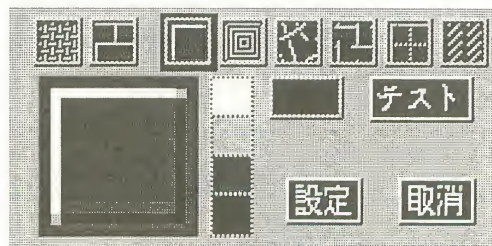
(1) プログラムの仕様

[OPT.1] と [OPT.2] を同時に押すと、その時点でアクティブなウィンドウの内容をハードコピーします。[CTRL] と [SHIFT]、そして [OPT.1] と [OPT.2] を同時に押すことで、プログラムは終了します。

印刷環境はデフォルトのものを使用しますので、あらかじめコントロールパネル等で設定しておいてください。また、カラーのプリンタを使用した場合でも、印刷はモノクロで行われます。

図 1 は、背景設定 .X をアクティブにした状態で PRNSMPL を利用した例です。

■図 1 PRNSMPL の実行例



(2) プログラムの説明

このプログラムでは、ハードコピーという、複雑かつ一定でない画像の印刷を行うわけですが、このような処理には自由度のもっとも高いプロセス印刷が適しています。

プロセス印刷の処理の要は、プリンタドライバに登録して使用するユーザープロセスにあります。プロセス印刷を行う手順自体は、印刷内容によらずほぼ一定で、PRNSMPL でも常識的な処理を行っています (PRNSMPL.S:45~48, 88~101, 125~135, 144 行)。PRNSMPL.S の 202~319 行が、問題のユーザープロセスです。

PRNSMPL のユーザープロセスが行っている仕事を簡条書きにしてみると、以下のようになります。

● 印刷すべき範囲の算出

ユーザープロセスが呼ばれた時点では、カレントビットマップとして、プリントマネージャが用意したビットマップがセットされています。すでに説明したように、いわば、このビットマップはプリンタの紙面そのものであり、ここに描画したビットパターンがそのままプリンタ用紙に印刷されます。このビットマップのビットマップレクタングルを調べて、現在印刷することが要求されているのはプリンタ用紙のどのあたりであるか、また、その場所にはどのような画像を印刷すべきかを判断します。

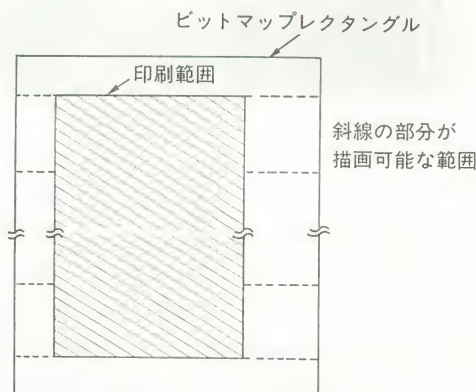
PRNSMPL では、あらかじめアイドルイベントの処理中で作成しておいたアクティブなウィンドウ内部をコピーしたビットの内容を印刷します。プリントマネージャから渡されたビットマップのビットマップレクタングルから、このビットのどのあたりを印刷すべきかを判断しています (PRNSMPL.S:205~246 行など)。

● クリッピング処理

プロセス印刷では、クリッピング処理はすべてユーザープロセスが行わなければならない。描画を制限する領域としては2つの矩形型の領域、ビットマップレクタングルと印刷範囲があり、その関係は図2のようになっています。

印刷範囲は、印刷環境設定ウィンドウによって設定されるレクタングルです。ビットマップレクタングルは、一般にその範囲よりも大きい、等しくなっています。結局、ユーザープロセス内で描画できるのは、ビットマップレクタングル内部で、かつ印刷範囲内部にかぎられます。印刷範囲の外にも描画は可能で、その場合はそのまま印刷されてしまうので注意してください。

■図2 クリッピングすべき領域



PRNSMPL では、この処理を 306~307 行で（若干簡略化していますが）行っています。これに加えて、もともとのイメージの右端の半端なドット（8 ドットに足りない端数）をマスクする処理を行っているため (PRNSMPL.S:278~282 行)、若干煩雑な印象を与え

ているかもしれません。

● 4 階調→タイリング処理

4 階調で表示されているものをモノクロで表現する場合は、タイリング等を行って疑似階調で表現するしかありません。PRNSMPL でも、かんたんなタイリング処理を行って濃淡を表現しています (PRNSMPL.S:284~300 行)。

● プリントマネージャのビットマップへのデータ書き込み

以上のようにして生成した印刷イメージは、カレントビットマップに格納します(302~303 行)。プリントマネージャは、ユーザープロセスから戻ってきた時点で、ビットマップに入っていたイメージをプリンタドライバに渡し、プリンタドライバはそれを各プリンタで印字できる形式に変換して出力することになります。

(3) プログラムリスト

リスト 1 に WORKING, リスト 2 に PRNSMPL.S を示します。また、リスト 3 に実行ファイルを作成するための makefile を示します。スケルトンは、標準のものをそのまま使用してください。

■ リスト 1 PRNSMPL 用 WORK.INC

```

1  *
2  *          SX-WINDOW
3  *          PRNSMPL
4  *
5  *          ワーク定義用インクルードファイル
6  *
7
8  STKSIZE      =          2*1024          * スタックサイズ
9
10 *          ワークの内容の定義
11
12             .offset 0
13 cmdLine:          * コマンドラインのアドレス
14             ds.l    1          * を保存するワーク
15 envPtr:           * 環境のアドレスを
16             ds.l    1          * 保存するワーク
17 winRect:         * ウィンドウ
18             ds.l    2          * レクタングルレコード
19 paramFlg:        * コマンドラインの
20             ds.w    1          * 解析結果を示す
21 eventRec:        * フラグ
22 eventRec_what:   * イベントレコードの先頭
23             ds.w    1          * イベントコード
24 eventRec_whom1:  * 第 1 引数
25             ds.l    1
26 eventRec_when:   * イベント発生時
27             ds.l    1
28 eventRec_whom2:  * 第 2 引数

```

```

29          ds. l      1
30 eventRec_what2:
31          ds. w      1          * タスクマネージャイベント
                                   * の種類
32 eventRec_taskID:
33          ds. w      1          * 送り手のタスクID
34 eventMask:          * イベントマスクを
                                   * 保存するワーク
35          ds. w      1
36 taskID:             * タスクIDを
                                   * 保存するワーク
37          ds. l      1
38
39 prnFlag:
40          ds. w      1          * 印刷中フラグ
41 prEvHdl:
42          ds. l      1          * 印刷環境レコードへの
                                   * ハンドル
43 bitsRect:
44          ds. l      2          * ビッツのレクタングル
45
46
47 WORKSIZE:          * ワークの終了
48

```

■リスト2 PRNSMPL.S

```

1  *
2  *          SX-WINDOW
3  *          PRNSMPL
4  *
5  *          初期化&終了&イベント処理モジュール
6  *
7
8          .include      DOSCALL.MAC
9          .include      SXCALL.MAC
10
11         .xdef      _INIT, _TINI
12         .xdef      IDLE
13         .xdef      MSLDOWN, MSLUP, MSRDOWN, MSRUP
14         .xdef      KEYDOWN, KEYUP
15         .xdef      UPDATE, ACTIVATE
16         .xdef      SYSTEM1, SYSTEM2, SYSTEM3, SYSTEM4
17
18         .include      WORK. INC          * ワークエリアの
                                           * 内容を定義するファイル
19
20         .text
21 MSLDOWN:          * [ レフトダウンイベント ]
22 MSLUP:            * [ レフトアップイベント ]
23 MSRDOWN:          * [ ライトダウンイベント ]
24 MSRUP:            * [ ライトアップイベント ]
25 KEYDOWN:          * [ キーダウンイベント ]
26 KEYUP:            * [ キーアップイベント ]
27 UPDATE:          * [ アップデートイベント ]
28 ACTIVATE:         * [ アクティベートイベント ]
29 SYSTEM3:          * [ システムイベント3 ]
30 SYSTEM4:          * [ システムイベント4 ]
31         moveq      #0, d0          * 以上のイベントでは
                                           * なにもしない
32
33         rts

```

```

34 IDLE:
35         tst.b    prnFlag (a5)
36         bne      Printing
37
38         move.l   eventRec_what2 (a5), d0
39
40         and.b    #$0f, d0
41
42         cmp.b    #$0f, d0
43         beq      KillMe
44         cmp.b    #$0c, d0
45         bne      IDLE9
46
47         move.w   #-1, -(sp)
48
49         SXCALL   $A4E2
50         addq.l   #2, sp
51         bmi      DriverBusy
52
53         SXCALL   $A20F
54         move.l   a0, a2
55
56         move.l   (a2), a1
57
58         tst.w    (a1)
59         bne      PrintEnd
60         move.l   4(a2), d0
61
62         move.l   8(a2), d1
63         sub.w    d0, d1
64         swap     d0
65         swap     d1
66         sub.w    d0, d1
67         swap     d1
68         clr.l    bitsRect (a5)
69         move.l   d1, bitsRect+4 (a5)
70
71         move.w   #%0011, -(sp)
72         pea      bitsRect (a5)
73         clr.w    -(sp)
74         SXCALL   $A1CA
75         addq.l   #8, sp
76         bmi      PrintEnd
77         move.l   a0, bitsHdl
78
79         pea      (a0)
80         SXCALL   $A1CC
81         addq.l   #4, sp
82
83         clr.l    -(sp)
84         clr.w    -(sp)
85         pea      bitsRect (a5)
86         pea      4(a2)
87         move.l   (a0), -(sp)
88         pea      (a1)
89         SXCALL   $A17F
90         lea      22(sp), sp
91
92         move.l   #$8e, -(sp)
93         SXCALL   $A021

```

```

# [ アイドルイベント ]
# 印刷中?
#  ならばPrintingへ

# シフトキービットを
# 取り出す
# OPT1 OPT2 CTRL SHIFT
# を取り出す
# 全部押されている?
#  ならばKillMeへ
# OPT1+OPT2が押されている?
#  でなければIDLE9へ
# 印刷開始!
# デフォルトの
# プリンタドライバを
# __PMOpenオープンする

# オープンできなければ
# DriverBusyへ

# _WMAActive
# アクティブウィンドウの
# レコードを得る
# ビットマップレコード

# テキストタイプ?
#  でなければPrintEndへ
# グラフポート
# レクタングル (左上)
#   " (右下)

# ビッツの大きさを意味する
# レクタングル

# 2ページ
# ビッツの大きさ
# テキストタイプ
# __GMNewBits ビッツを作成

# エラーならPrintEndへ
# ビッツへのハンドルを
# 静的ワークに収める
# (ユーザープロセスは
# 動的ワークエリアを
# アクセスできない)
# ビッツをロック
# __GMLockBits

# マスクなし
# コピーモード PSET
# コピー先レクタングル
# コピー元レクタングル
# コピー先ビットマップ
# コピー元ビットマップ
# __GMCopy

# 印刷環境レコードのサイズ
# __MMChHdlNew

```

```

90      addq. l    #4, sp
91      move. l    d0, prEvHdl (a5)      * ブロックへの
                                           * ハンドルを保存
                                           * 作成したブロック中に
92      move. l    d0, -(sp)              * PMSetDefault
93      SXCALL     $A4E4                  * デフォルトの印刷環境を構築

94      addq. l    #4, sp
95
96      pea        userProc (pc)          * ユーザープロセスを指定して
97      move. l    prEvHdl (a5), -(sp)    * デフォルトの印刷環境で
98      SXCALL     $A4FA                  * PMProcPrintプロセス印刷
                                           * 開始

99      addq. l    #8, sp
100
101      st         prnFlag (a5)           * 印刷中フラグを立てる
102 IDLE9:
103      moveq      #0, d0
104      rts
105
106 KillMe:
107      pea        quitMsg (pc)           * 終了確認メッセージ
108      move. w    #$004, -(sp)          * はい/いいえ
109      SXCALL     $A2F6                  * __DLError
110      addq. l    #6, sp
111      cmp. w     #2, d0
112      beq        IDLE9                  * 「いいえ」?
                                           *   ならばIDLE9へ
113
114      moveq      #-1, d0
115      rts
116
117 DriverBusy:
118      pea        busyMsg (pc)           * 印刷不能メッセージ
119      move. w    #$001, -(sp)          * 確認
120      SXCALL     $A2F6                  * __DLError
121      addq. l    #6, sp
122      bra        IDLE9
123
124 Printing:
125      clr. w     -(sp)                  * 印刷続行
126      SXCALL     $A4EF                  * __PMAction
127      addq. l    #2, sp
128      cmp. l     #1, d0
129      beq        IDLE9                  * 印刷中?
                                           *   ならばIDLE9へ
130 PrintEnd:
131      SXCALL     $A4E3                  * __PMCclose
132
133      move. l    prEvHdl (a5), -(sp)    * 印刷環境レコードを廃棄
134      SXCALL     $A038                  * __MMHdlDispose
135      addq. l    #4, sp
136
137      move. l    bitsHdl, -(sp)         * ビッツを廃棄
138      SXCALL     $A1CB                  * __GMDisposeBits
139      addq. l    #4, sp
140
141      clr. l     prEvHdl (a5)
142      clr. l     bitsHdl
143
144      sf         prnFlag (a5)           * 印刷中フラグをおろす
145
146      moveq      #0, d0
147      rts
148
149
150 SYSTEMI:
                                           * [ システムイベント1 ]

```



```

151 SYSTEM2:
152     move.w    eventRec_what2(a5), d0      * [ システムイベント2 ]
153     cmp.w     #1, d0                      * タスクの終了?
154     beq        AllClose                   *   ならばLetsGoAwayへ
155     cmp.w     #2, d0                      *   全ウィンドウのクローズ?
156     beq        AllClose                   *   ならばLetsGoAwayへ
157
158     moveq     #0, d0
159     rts
160
161 AllClose:
162     moveq     #-1, d0
163     rts
164
165 _INIT:
166                                     * [ アプリケーションの
167                                     *   初期化を行なう ]
168     sf        prnFlag(a5)
169     clr.l     prEvHdl(a5)
170     clr.l     bitsHdl
171
172     tst.l     d1
173     bne        _INIT_Err                 * 2個目の起動?
174                                     *   ならば_INIT_Errへ
175
176     SXCALL    $A360                      * _TSGetID
177     move.l     d0, taskID(a5)            * タスクIDを得る
178
179     moveq     #0, d0
180     rts
181
182 _INIT_Err:
183     moveq     #-1, d0
184     rts
185
186 _TINI:
187     tst.l     prEvHdl(a5)                * [ 終了処理 ]
188                                     * 印刷環境レコードが
189                                     *   作成されたまま?
190     beq        _TINIO                     *   でなければ_TINIOへ
191
192     move.l     prEvHdl(a5), -(sp)          * 印刷環境レコードを
193                                     *   廃棄する
194     SXCALL    $A038                      * _MMHdlDispose
195     addq.l     #4, sp
196
197 _TINIO:
198     tst.l     bitsHdl
199     beq        _TINI1                     * ビッツが作成されたまま?
200                                     *   でなければ_TINI1へ
201
202     move.l     bitsHdl, -(sp)              * ビッツを廃棄する
203     SXCALL    $A1CB                      * _GMDisposeBits
204     addq.l     #4, sp
205
206 _TINI1:
207     moveq     #0, d0
208     rts
209
210 userProc:
211                                     * [ プロセス印刷用
212                                     *   ユーザープロセス ]
213     move.l     4(sp), a3                  * 印刷環境レコードへの
214                                     *   ハンドル
215     move.l     8(sp), a4                  * 描画範囲のレクタングル
216                                     *   へのポインタ
217     move.l     bitsHdl, a0                * ビッツへのハンドル
218     move.l     (a0), a6                  * A6: ビッツへのポインタ
219
220     SXCALL    $A132                      * _GMGetGraph

```

209	move. l	(a0), a1	* ビットマップへのポインタ
210	move. w	4(a1), d1	* D1 : Y先頭
211	move. w	8(a1), d2	* Y終端
212	move. w	6(a4), d0	* 描画範囲のY終端
213	cmp. w	d2, d0	* 描画範囲の方が広い?
214	bge	userProc0	* ならばuserProc0へ
215			
216	move. l	d0, d2	
217	userProc0:		
218	move. w	8(a6), d0	* ビットマップY終端
219	add. w	d0, d0	* ×2
220	cmp. w	d2, d0	* ビットマップの方が広い?
221	bcc	userProc1	* ならばuserProc1へ
222			
223	move. l	d0, d2	
224	userProc1:		
225	sub. w	d1, d2	
226	asr. w	#1, d2	
227	subq. w	#1, d2	* D2 : ライン数 ÷ 2 - 1
228			
229	move. w	6(a1), d0	* (X終端
230	sub. w	2(a1), d0	* -X先頭)
231	move. w	14(a1), a4	* A4 : 1ラインのバイト数
232	move. l	10(a1), a1	* A1 : ビットマップの * ベースアドレス
233			* 半端なドット数
234	and. w	#15, d0	
235	move. w	#\$fff, d3	
236	lsl. w	d0, d3	* D3 : 右端の1ワード用マスク
237	not. w	d3	
238			* ビットイメージへのポインタ
239	move. l	10(a6), a2	
240	move. w	d1, d0	* ビットマップのY先頭 ÷ 2
241	lsl. w	#1, d0	* A5 : ビットの
242	move. w	14(a6), a5	* 1ラインのバイト数
243	mulu	14(a6), d0	
244	lea	(a2, d0. l), a2	* A2 : ビットを読み出す * 先頭アドレス (page 0)
245	move. l	16(a6), d0	
246	lea	(a2, d0. l), a3	* A3 : ビットを読み出す * 先頭アドレス (page 1)
247	move. w	6(a6), d0	* ビットの横ドット数
248	and. w	#7, d0	* 半端なドット数
249	move. b	#\$ff, d4	
250	lsl. w	d0, d4	
251	not. b	d4	* D4 : ビットの右端の * 1バイト用マスク
252			
253	move. w	6(a6), d1	* ビットの1ラインのドット数
254	lsl. w	#3-1, d1	* ÷ 8 × 2
255	cmp. w	d1, a4	* ビットの横 ≤
256	bcc	userProc10	* ビットマップの横?
257			* ならばuserProc10へ
258	move. l	a4, d1	
259	lsl. w	#1, d1	* ビットの横
260	move. b	#\$ff, d4	* = ビットマップの横 ÷ 2
261	bra	userProc2	
262	userProc10:		
263	lsl. w	#1, d1	
264	addq	#1, d1	
265	userProc2:		
266	tst. w	d2	* 描画すべき範囲がない?
267	bmi	userProc9	* ならばuserProc9へ

```

268 userProc3:
269         move.w    d1, d7                * ビットマップの
                                           * 1ラインのバイト数
                                           *   -1→カウンタ
270         subq.w    #1, d7
271         movem.l   d1/a1-a3, -(sp)
272 userProc4:
273         moveq     #0, d0                * D0 : データ作成用
                                           * (even line)
274         moveq     #0, d1                * D1 : データ作成用
                                           * (odd line)
275         move.b    (a2)+, d5             * D5 : ページ0
276         move.b    (a3)+, d6             * D6 : ページ1
277
278         tst.w     d7                    * 最後のバイト?
279         bne       userProc5             *   でなければuserProc5へ
280         and.w     d4, d5                *   不要なビットをマスク
281         and.w     d4, d6                *   不要なビットをマスク
282 userProc5:
283         swap      d7
284         move.w    #8-1, d7
285 userProc6:
286         roxl.b    #1, d5                * タイルパターンを
287         roxl.w    #1, d0                *   作成する
288         roxl.b    #1, d6                *   (イージーな方法ではある)
289         bcs       userProc7
290
291         roxl.w    #1, d0                * even line
292         lsl.w     #2, d1                * odd line
293         bra       userProc8
294 userProc7:
295         roxl.w    #1, d0                * even line
296         rol.w     #2, d1                * odd line
297         or.w      #3, d1
298 userProc8:
299         dbra      d7, userProc6
300
301         swap      d7
302         move.w    d0, (a1)+             * プリンタドライバの
                                           * ビットマップへ
                                           *   "
303         move.w    d1, -2(a1, a4)
304
305         dbra      d7, userProc4         * 1ライン全部に
                                           *   書き込むまでループ
306         and.w     d3, -2(a1)            *   右端の不要な部分をマスク
                                           *   (even line)
307         and.w     d3, -2(a1, a4)        *   "   (odd line)
308
309         movem.l   (sp)+, d1/a1-a3
310
311         lea       (a1, a4), a1          * ビットマップ
312         lea       (a1, a4), a1          *   2行下に
313         lea       (a2, a5), a2          * ビッツページ0次の行へ
314         lea       (a3, a5), a3          * ビッツページ1次の行へ
315
316         dbra      d2, userProc3         * すべてのラインに
                                           *   書き込むまでループ
317 userProc9:
318         moveq     #0, d0
319         rts
320
321         .even                                           * [ 固定データ ]
322 quitMsg:
323         dc.b      ' PRNSMPL:', 13
324         dc.b      ' プログラムを終了します.', 13

```

```

325          dc, b    ' よろしいですか?', 0
326 busyMsg:
327          dc, b    ' PRNSMPL:', 13
328          dc, b    ' 他のタスクが印刷中です.', 0
329
330          . even
331 bitsHdl:
332          ds, l     1                                * ビッツへのハンドル
333
334          . end

```

■リスト3 PRNSMPL 用 makefile

```

# PRNSMPL用 makefile

PRNSMPL. X:      SKELTON. o PRNSMPL. o
               lk -oPRNSMPL SKELTON PRNSMPL

SKELTON. o:      SKELTON. s WORK. INC
               as SKELTON

PRNSMPL. o:      PRNSMPL. s WORK. INC
               as PRNSMPL

```

2 サブウィンドウマネージャのサンプルプログラム

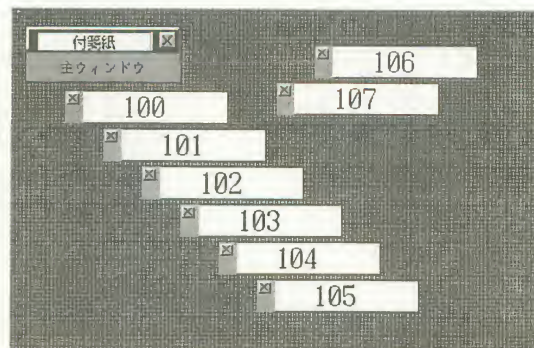
サブウィンドウマネージャのサンプルプログラムは、サブウィンドウを利用して画面上に付箋紙（いわゆるポスト・イット*1）型のウィンドウを表示するプログラムです。

*1:ポスト・イットは米国 3M 社の商標です。

(1) プログラムの仕様

SWSMPL を起動すると、画面中に主ウィンドウと、8 枚の付箋紙ウィンドウが表示されます（図3）。

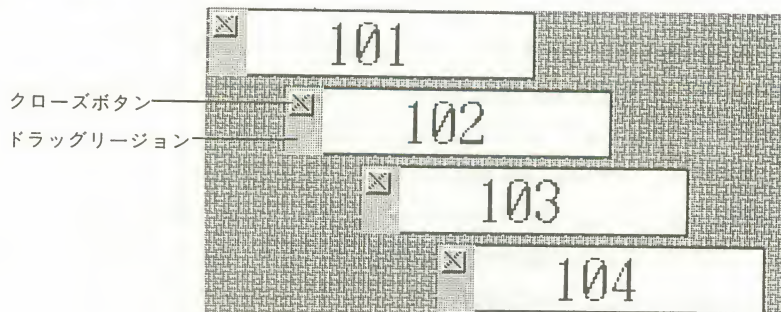
■図3 SWSMPL の実行例



主ウィンドウについては特筆するほどの機能は用意されていません。ドラッグとクローズのみ可能です。

付箋紙ウィンドウの内部には、それぞれのプライオリティが表示されています。それぞれの付箋紙には赤いドラッグリージョンとクローズボタンが備えられていて、ドラッグとクローズが可能です (図 4)。

■図 4 付箋紙ウィンドウ



主ウィンドウがインアクティブになる際のサブウィンドウの動き、プライオリティによるサブウィンドウ間の重なりぐあい等を確認してください。

なお、このプログラムは一度に 1 つしか実行できず、2 個目以降を起動させようとした場合、1 個目をアクティブにして終了します。

(2) プログラムの説明

このプログラムのポイントとなる部分は、マウスの左ボタンによるサブウィンドウの操作 (SWSMPL.S:70~199 行)、複数のウィンドウをもつ場合のアップデート処理 (SWSMPL.S:201~240 行)、そして、サブウィンドウをもつ場合のアクティベート処理 (SWSMPL.S:242~270 行) です。

・マウスの左ボタンによるサブウィンドウの操作

普通のウィンドウならば `SXCallWindM` などでも処理できることを、サブウィンドウの場合はすべて自分で行わなければなりません。このプログラムでは、ドラッグとクローズボタンの処理を行っています。実際の処理を行っているのは 88~199 行のサブルーチン `SubWinProcess` で、70~86 行では、どのサブウィンドウに対して処理を行うべきかを調べています。

`SubWinProcess` では、まずマウスが押されたポイントがサブウィンドウ内のどの部分に相当するかを調べ、それぞれの部分に対応した処理に分岐します (95~110 行)。

112~160 行ではサブウィンドウのドラッグ処理を行っています。`$A205 WMDrag` はウィンドウ定義関数を呼び出しているため使用できません。かわりに、`$A225 WMDragRgn`

を利用してサブウィンドウのアウトサイドリージョンをマウスに追従して動かしています。WMDragRgn は、少しも動かされなかった場合、移動させるリージョンを空にしてしまうので、作業用リージョンにアウトサイドリージョンをコピーして、それを移動させることにします (SWSMPL.S:121~124 行)。

移動後、\$A22B WSDelist を読んで、いったんサブウィンドウを画面から消し、レコード内の数値の操作を開始します (SWSMPL.S:138~140 行)。WMDragRgn が返した移動量を示すポイントにしたがって、アウトサイドリージョン、アップデートリージョン、グラフポートレクタングルをスライドさせることで、WMMove と同様な結果が得られます (SWSMPL.S:146~156 行)。その後、再度サブウィンドウを表示させることで、新しい場所にサブウィンドウが開かれます (SWSMPL.S:158~160 行)。

165~199 行ではクローズボタンの処理を行っています。その内容は、マウスの左ボタンが離されるまで座標を追いつづけ、ポインタがクローズボタン上にある場合は反転した状態のボタンを、クローズボタン上にない場合は通常の状態のボタンを、\$A182 GMPlotImg を使って描画します。マウスの左ボタンが最終的にクローズボタンの上で離されたなら、\$A229 WSDispose でサブウィンドウをクローズします。

● 複数のウィンドウをもつ場合のアップデート処理

ウィンドウを 1 つしかもたないタスクの場合、自分のではないウィンドウを対象にアップデートイベントが発生した場合に書き換えを行っても、それほど時間のロスにはなりません。が、複数のウィンドウをもつ場合、どのウィンドウをアップデートすべきかを判断したほうがスピード的に有利です。

207~213 行では主ウィンドウのアップデートを行っています。202~205 行の判断で、主ウィンドウをアップデートすべきである場合に、この処理を行っています。207 行で引数をスタックに積んで、\$A20D WMUpdate を呼んだ直後にスタックポインタの補正を行っています。これは 212 行で呼んでいる \$A20E WMUpdtOver で流用するためです。全体としてスタックポインタの値に矛盾は生じないので問題はありませんが、スタックの内容を変えてしまうような処理が間に入る場合には注意が必要です。

216 行からはサブウィンドウのアップデート処理です。

まず、主ウィンドウがアクティブかどうかを判断します (SWSMPL.S:216~217 行)。主ウィンドウがインアクティブの場合、サブウィンドウは表示されているはずがないので、何も行いません。アクティブならば、普通のウィンドウ同様のアップデート処理を行います (SWSMPL.S:223~237 行)。

● サブウィンドウをもつ場合のアクティベート処理

サブウィンドウを使用するタスクは、アクティベートイベントが発生し、主ウィンドウがアクティブとなったと判断できた場合、サブウィンドウをサブウィンドウリストに追加する処理

を行わなければなりません。235～262 行で、この処理を行っています。

(3) プログラムリスト

リスト 4 に WORK.INC, リスト 5 に SWSMPL.S を示します。また、リスト 6 に実行ファイルを作成するための makefile を示します。スケルトンは標準のものをそのまま使用してください。

■リスト 4 SWSMPL 用 WORK.INC

```

1 *
2 *           SX-WINDOW
3 *           SWSMPL
4 *
5 *           ワーク定義用インクルードファイル
6 *
7
8 STKSIZE      =      2*1024           * スタックサイズ
9
10 *           ワークの内容の定義
11
12             .offset 0
13 cmdLine:                                * コマンドラインの
14                                     * アドレスを
15 envPtr:      ds. l      1             * 保存するワーク
16                                     * 環境のアドレスを保存
17                                     * するワーク
18
19 paramFlg:    ds. l      1
20
21 eventRec:    ds. w      1             * コマンドラインの
22 eventRec_what:                * 解析結果を示す
23                                     * フラグ
24 eventRec_whom1:                * イベントレコードの先頭
25                                     * イベントコード
26                                     * 第 1 引数
27 eventRec_when:                * イベント発生時
28 eventRec_whom2:                * 第 2 引数
29                                     *
30 eventRec_what2:                *
31                                     * タスクマネージャ
32                                     * イベントの種類
33 eventRec_taskID:                *
34 eventMask:    ds. w      1             * 送り手のタスクID
35                                     * イベントマスクを
36                                     * 保存するワーク
37
38 taskID:      ds. w      1             * タスクIDを保存するワーク
39
40 winPtr:      ds. l      1             * ウィンドウレコードを
41                                     * 作成する場所
42
43 winActive:    ds. b      $72
44
45 rgnHdl:      ds. w      1             * アクティブフラグ
46
47                                     * 作業用リージョンへの
48                                     * ハンドル

```

```

44          ds.l    1
45 dragRect:
46          ds.l    2
47
48 sWinPtrList:
49          ds.l    8
50
51 WORKSIZE:
52

```

* ドラッグ可能領域を
 * 意味する
 * レクタングルレコード
 * サブウィンドウレコード
 * へのポインタのリスト
 * ワークの終了

■リスト5 SWSMPL.S

```

1  *
2  *      SX-WINDOW
3  *      SWSMPL
4  *
5  *      初期化&終了&イベント処理モジュール
6  *
7
8      .include      DOSCALL.MAC
9      .include      SXCALL.MAC
10
11     .xdef      _INIT, _TINI
12     .xdef      IDLE
13     .xdef      MSLDOWN, MSLUP, MSRDOWN, MSRUP
14     .xdef      KEYDOWN, KEYUP
15     .xdef      UPDATE, ACTIVATE
16     .xdef      SYSTEM1, SYSTEM2, SYSTEM3, SYSTEM4
17
18     .include      WORK.INC      * ワークエリアの内容を
19                                * 定義するファイル
20
21 WINOPT      =      %0000      * ウィンドウオプション
22 WIN_X       =      128        * ウィンドウ初期 x
23 WIN_Y       =      24         * ウィンドウ初期 y
24 SWIN_X      =      120+16     * サブウィンドウ初期 x
25 SWIN_Y      =      28         * サブウィンドウ初期 y
26 SWIN        =      8          * サブウィンドウ個数
27
28 .text
29 IDLE:
30 MSLUP:
31 MSRDOWN:
32 MSRUP:
33 KEYDOWN:
34 KEYUP:
35 SYSTEM3:
36 SYSTEM4:
37     moveq     #0, d0
38
39     rts
40
41 MSLDOWN:
42     move.l    eventRec_whoml(a5), a0
43     lea       winPtr(a5), a2
44     cmp.l     a2, a0
45     bne       MS�D_SUB
46
47     tst.b     winActive(a5)

```

* [アイドルイベント]
 * [レフトアップイベント]
 * [ライトダウンイベント]
 * [ライトアップイベント]
 * [キーダウンイベント]
 * [キーアップイベント]
 * [システムイベント3]
 * [システムイベント4]
 * 以上のイベントでは
 * なにもしない
 * [レフトダウンイベント]
 * 自分のウィンドウ上で
 * 発生したか?
 * 違うならMSLD_SUBへ
 * 現在ウィンドウは
 * アクティブか?


```

47      bne     MSLD1          # アクティブならMSLD1へ
48      pea     (a2)
49      SXCALL  $A1FE          # _WMSelect
50      addq.l   #4, sp
51      bra     MSLD9          # アクティブにするだけ
52 MSLD1:
53      pea     eventRec(a5)
54      pea     (a2)           # ウィンドウ処理
55      SXCALL  $A3A2          # _SXCallWindM
56      addq.l   #8, sp
57      tst.l    d0            # どこも操作されなかった?
58      beq     MSLD9          #   ならばMSLD9へ
59
60      cmp.w    #7, d0         # クローズボタン?
61      beq     CloseBttn      #   ならばCloseBttnへ
62 MSLD9:
63      moveq    #0, d0
64      rts
65
66 CloseBttn:
67      moveq    #-1, d0
68      rts
69
70 MSLD_SUB:
71      lea     sWinPtrList(a5), a4 # リストの先頭アドレス
72      moveq    #SWIN-1, d7
73 MSLD_SUB0:
74      move.l   (a4), d1      # サブウィンドウ
75                                     # レコードへのポインタ
76      beq     MSLD_SUB1      # クローズされているなら
77                                     # MSLD_SUB1へ
78      cmp.l    eventRec_whom1(a5), d1 # 押されたのは
79                                     # このサブウィンドウの上?
80      bne     MSLD_SUB1      #   でなければMSLD_SUB1へ
81
82      bsr     SubWinProcess   # サブウィンドウ処理
83      move.l   d0, (a4)       # 結果をリストに戻す
84
85 MSLD_SUB1:
86      addq.l   #4, a4          # 次のサブウィンドウへ
87      dbra     d7, MSLD_SUB0 #   個数分繰り返す
88
89      moveq    #0, d0
90      rts
91
92 SubWinProcess:
93      move.l   d1, a2          # [ サブウィンドウ処理 ]
94                                     # サブウィンドウ
95                                     # レコードへのポインタ
96
97      pea     (a2)           # サブウィンドウ内部
98      SXCALL  $A131          # _GMSetGraph
99      addq.l   #4, sp
100
101      move.l   eventRec_whom2(a5), -(sp) # マウスが押されたポイントを
102      SXCALL  $A13F          #   _GMGlobalToLocal
103      addq.l   #4, sp        #   でローカル座標系に変換
104      move.l   d0, d1
105
106      move.l   d1, -(sp)
107      pea     SWDragRect(pc) # ドラッグリージョンの上?
108      SXCALL  $A156          #   _GMPtInRect
109      addq.l   #8, sp        #   でなければ
110      beq     SWProc9        #   SWProc9へ

```

106	move.l	d1, -(sp)	
107	pea	SWCBoxRect (pc)	* クローズボタンの上?
108	SXCALL	\$A156	* __GMPtInRect
109	addq.l	#8, sp	* ならば
110	bne	SWCcloseBox	* SXCloseBoxへ
111			
112	SXCALL	\$A210	* __VMGraphGet
113	pea	(a0)	* ウィンドウマネージャの * グラフポート
114	SXCALL	\$A131	* __GMSetGraph
115	addq.l	#4, sp	
116			
117	move.w	#%0011, -(sp)	* 2 ページアクティブで * ラバーバンドを描画
118	SXCALL	\$A149	* __GMAPage
119	addq.l	#2, sp	
120			
121	move.l	\$48 (a2), -(sp)	* アウトサイドリージョンを
122	move.l	rgnHdl (a5), -(sp)	* 作業用リージョンにコピー
123	SXCALL	\$A160	* __GMCopyRgn
124	addq.l	#8, sp	
125			
126	clr.l	-(sp)	* ユーザープロセスは無し
127	move.w	#3, -(sp)	* 上下左右にドラッグ許可
128	pea	dragRect (a5)	* ドラッグ可能領域
129	pea	dragRect (a5)	* ドラッグ可能領域
130	move.l	eventRec_whom? (a5), -(sp)	* ドラッグ開始ポイント
131	move.l	rgnHdl (a5), -(sp)	* インサイドリージョン
132	SXCALL	\$A225	* __WMDragRgn
133	lea	22 (sp), sp	
134	move.l	d0, d1	
135	tst.l	d0	* 少しでも動いた?
136	beq	SWProc9	* 動いていなければSWProc9へ
137			
138	pea	(a2)	* 一旦リストから外して * 表示停止
139	SXCALL	\$A22B	* __WSDeList
140	addq.l	#4, sp	
141			
142	pea	(a2)	* サブウィンドウレコードの * 値を操作開始
143	SXCALL	\$A131	* __GMSetGraph
144	addq.l	#4, sp	
145			
146	move.l	d1, -(sp)	
147	move.l	\$48 (a2), -(sp)	* アウトサイドリージョンを
148	SXCALL	\$A162	* __GMSlideRgn
149	addq.l	#8, sp	* でスライド
150	move.l	d1, -(sp)	
151	move.l	\$50 (a2), -(sp)	* アップデイトリージョンを
152	SXCALL	\$A162	* __GMSlideRgn
153	addq.l	#8, sp	* でスライド
154	move.l	d1, -(sp)	* グラフポートレクタングルを
155	SXCALL	\$A137	* __GMSlideGraph
156	addq.l	#4, sp	* でスライド
157			* 以上でWMMove相当の * 処理が完了
158	pea	(a2)	* 新しい位置でリストに * 加えて再表示
159	SXCALL	\$A22A	* __WSEnList
160	addq.l	#4, sp	
161	SWProc9:		
162	move.l	a2, d0	* 返り値は * サブウィンドウレコード

```

163          rts
164
165 SWCloseBox:
166          moveq    #1, d1
167          bra      SWCBox0
168
169 SWCBoxLoop:
170          SXCALL   $A0AC
171          tst.l    d0
172          beq      SWCBoxRel
173
174          SXCALL   $A0A7
175          move.l   d0, -(sp)
176          pea      SWCBoxRect(pc)
177          SXCALL   $A156
178          addq.l   #8, sp
179          cmp.l    d0, d1
180          beq      SWCBoxLoop
181
182          move.l   d0, d1
183          SXCALL   $A06C
184
185          move.w   d1, -(sp)
186          pea      SWCBoxRect(pc)
187          pea      SWCBoxPImg(pc)
188          SXCALL   $A182
189          lea      10(sp), sp
190          SXCALL   $A06B
191
192          bra      SWCBoxLoop
193
194 SWCBoxRel:
195          tst.l    d1
196          beq      SWProc9
197
198          pea      (a2)
199          SXCALL   $A229
200          addq.l   #4, sp
201
202          moveq    #0, d0
203          rts
204
205 UPDATE:
206          move.l   eventRec_whom1(a5), a1
207          lea      winPtr(a5), a2
208          cmp.l    a1, a2
209          bne      UPDATE0
210
211          pea      (a2)
212          SXCALL   $A20D
213
214          bsr      DrawGraphMain
215
216          SXCALL   $A20E
217          addq.l   #4, sp
218          bra      UPDATE0
219
220          SXCALL   $A20E
221          addq.l   #4, sp
222          bra      UPDATE0
223
224          tst.b     winActive(a5)
225          beq      UPDATE0

```

* [クローズボタンの処理]
 * 最初は押されている状態
 * まずボタンの状態を
 * 描画させる
 *
 * _EMLWait
 * 左ボタンは離された?
 * ならばSXCBoxRelへ
 *
 * _EMMSLoc
 * 現在のマウスの位置
 * クローズボタン中?
 * _GMPtInRect
 *
 * 前と状態が変わっている?
 * 変わっていなければ
 * SXCBoxLoopへ
 *
 * _MSHideCsr
 * マウスポインタを消して
 * 現在のボタンの状態に
 * 合わせて
 * クローズボタンの位置
 * クローズボタンの
 * プロットイメージ
 * _GMPlotImg
 * で描画
 * _MSShowCsr
 * マウスポインタを再表示
 * 離されるまでループを
 * 繰り返す
 *
 * 結局ボタン上で離された?
 * 違うのならSWProc9へ
 *
 * サブウィンドウを廃棄
 * _WSDispose
 *
 * 返り値で廃棄したことを通知
 *
 * [アップデートイベント]
 *
 * 自分のウィンドウ上で
 * 発生したか?
 * 違うならばUPDATE0へ
 *
 * ウィンドウを
 * _WMUpdate
 * でアップデート開始
 * ウィンドウ内部を描画
 * 前にスタックに積んだ
 * 値を利用して
 * _WMUpdtOver
 * アップデート終了
 *
 * ウィンドウはアクティブ?
 * でなければサブウィンドウは

```

218                                     * 表示されていないので
219                                     *   UPDATE9へ
220     lea      sWinPtrList(a5), a4    * リストの先頭
221     moveq    #0, d2                 * サブウィンドウ番号
222     moveq    #SWIN-1, d7
223 UPDATE1:
224     move.l   (a4)+, d1              * サブウィンドウレコードへの
225                                     * ポインタ
226     beq      UPDATE2               * クローズされているなら
227                                     * UPDATE2へ
228     cmp.l    a1, d1                * アップデートするのは
229                                     * このサブウィンドウ?
230     bne      UPDATE2               * 違うならばUPDATE2へ
231     move.l   d1, -(sp)
232     SXCALL   $A20D
233                                     * このサブウィンドウを
234                                     *   WMUpdate
235                                     * アップデート開始
236     bsr      DrawGraphSub           * サブウィンドウ内部を描画
237                                     * 前にスタックに積んだ
238                                     * 値を利用して
239     SXCALL   $A20E
240     addq.l   #4, sp                *   WMUpdtOver
241                                     * アップデート終了
242 UPDATE2:
243     addq     #1, d2
244     dbra     d7, UPDATE1           * サブウィンドウ番号+1
245                                     * 全部調べるまでループ
246 UPDATE9:
247     moveq    #0, d0
248     rts
249
250 ACTIVATE:
251     move.l   eventRec_whom1(a5), d0 * [ アクティベイトイベント ]
252     beq      ACT9
253     lea      winPtr(a5), a0
254     cmp.l    a0, d0
255     bne      ACT0
256     tst.b    winActive(a5)
257     bne      ACT9
258
259     st       winActive(a5)          * アクティブフラグをセット
260
261     lea      sWinPtrList(a5), a4    * リストの先頭
262     moveq    #SWIN-1, d7
263 ACT1:
264     move.l   (a4)+, d0              * サブウィンドウレコードへの
265                                     * ポインタ
266     beq      ACT2                   * クローズされているなら
267                                     * ACT2へ
268     move.l   d0, -(sp)
269     SXCALL   $A22A
270     addq.l   #4, sp
271                                     * このサブウィンドウを
272                                     *   WSEnlist
273                                     * でリストに加える
274     dbra     d7, ACT1               * 繰り返し
275
276     bra      ACT9
277
278 ACT0:
279     sf       winActive(a5)          * アクティブフラグを
280                                     * リセット
281
282 ACT9:
283     moveq    #0, d0
284     rts
285
286 SYSTEM1:
287 SYSTEM2:
288     move.w    eventRec_what2(a5), d0 * [ システムイベント1 ]
289                                     * [ システムイベント2 ]

```



```

274      cmp.w    #1, d0      * タスクの終了?
275      beq      A1IClose    *   ならばLetsGoAwayへ
276      cmp.w    #2, d0      * 全ウィンドウのクローズ?
277      beq      A1IClose    *   ならばLetsGoAwayへ
278      cmp.w    #$20, d0    * ウィンドウのセレクト?
279      beq      WindowSelect *   ならばWindowSelectへ
280
281      moveq     #0, d0
282      rts
283
284 A1IClose:
285      moveq     #-1, d0
286      rts
287
288 WindowSelect:
289      pea      winPtr (a5)    * 自分のウィンドウを
                                * セレクトする
                                * __WMSelect
290      SXCALL    $A1FE
291      addq.l    #4, sp
292
293      moveq     #0, d0
294      rts
295
296 _INIT:
                                * [ アプリケーションの
                                * 初期化を行なう ]
297      tst.l     d1
298      bne      _INIT_reEnter  * 2度目の実行?
                                *   ならばなにもせずに終了
299
300      move.l     winRect (a5), d0
301      move.w     paramFlg (a5), d1
302      btst      #0, d1
                                * '-W オプションが
                                * 指定された?
                                * 指定されていないければ
                                * _INIT0へ
303      beq      _INIT0
304
305      move.l     winRect+4 (a5), d1
306      beq      _INIT1
307      tst.w     d1
308      cmp.w     d0, d1
309      ble      _INIT1
310      swap      d0
311      swap      d1
312      cmp.w     d0, d1
313      bgt      _INIT2
314      swap      d0
315      swap      d1
316      bra      _INIT1
317 _INIT0:
318      SXCALL    $A35E
319      move.l     d0, winRect (a5)
                                * __TGetWindowPos
                                * デフォルト位置を得る
320 _INIT1:
321      add.l     #WIN_X*10000+WIN_Y, d0
                                * ウィンドウレクタングルを
                                * 作成
322      move.l     d0, winRect+4 (a5)
323 _INIT2:
324      SXCALL    $A431
                                * __TGetGrapMode
                                * デスクトップのサイズを
                                * 求める
                                * 画面モードは
                                * 4より小さい?
                                *   ならば_INIT4へ
325
326      cmp.w     #4, d0
                                * 画面モードは16以上?
                                *   ならば_INIT4へ
327      bcs      _INIT4
328      cmp.w     #16, d0
329      bcc      _INIT4
330      move.l     #512*10000+512, d0
                                * デスクトップサイズは

```

```

331                                     # 512×512
332 _INIT4:      bra      _INIT5
333      move.l    #1024*10000+1024, d0    # デスクトップサイズは
                                     # 1024×1024
334 _INIT5:
335      move.w    #(-SWIN_X), dragRect(a5) # ドラッグ可能領域の
                                     # 左上の座標 x
336      move.w    #(-SWIN_Y), dragRect+2(a5) # " y
337      move.l    d0, dragRect+4(a5)      # " 右下の座標
338
339      SXCALL    $A360                    # __TGetID
340      move.l    d0, taskID(a5)          # タスクIDを得る
341
342      move.l    d0, -(sp)                # タスクID
343      move.w    #-1, -(sp)               # クローズボタンあり
344      move.l    #-1, -(sp)               # もっとも手前に
345      move.w    #$31*16+WINOPT, -(sp)    # 標準ウィンドウ
                                     # (クローズボタンのみ)
346      move.w    #-1, -(sp)               # 可視
347      pea.l     winTitle(pc)             # ウィンドウタイトル
348      pea.l     winRect(a5)              # ウィンドウレクタングル
349      pea       winPtr(a5)               # ワーク上に作成
350      SXCALL    $A1F9                    # __WMOpen
351      lea.l     26(sp), sp               # ウィンドウを開く
352      tst.l     d0                       # エラー?
353      bmi       _INIT_Err                # ならば_INIT_Errへ
354
355      st         winActive(a5)            # アクティブフラグをセット
356
357      bsr        DrawGraphlst             # ウィンドウ内部を描画する
358      # (最初の1回)
359
360      moveq     #0, d0
361      rts
362 _INIT_Err:
363      moveq     #-1, d0                  # 初期化できなかった
364      rts
365 _INIT_reEnter:
366      SXCALL    $A360                    # __TGetID
367      move.w    d0, msgRec_Tskid
368
369      pea       msgRec(pc)                # WINDOWSELECTの
370      # メッセージレコード
371      move.w    d1, -(sp)                 # 親のタスクに送信
372      SXCALL    $A418                    # __TSSendMes
373      addq.l    #6, sp
374
375      moveq     #-2, d0                  # 初期化しなかった
376      rts
377
378      DrawGraphlst:
379      # ウィンドウ内部の描画の
380      # 準備をするサブルーチン
381      SXCALL    $A15A                    # __GMNewRgn
382      move.l    a0, rgnHdl(a5)
383
384      pea       SWRect(pc)                # サブウィンドウの大きさに
385      pea       (a0)                      # リージョンを設定
386      SXCALL    $A15F                    # __GMRectRgn
387      addq.l    #8, sp
388
389      lea       sWinPtrList(a5), a4      # リストの先頭
390      moveq     #100+SWIN, d6            # ブライオリティ
391      moveq     #SWIN-1, d7
392      DGLst0:

```

```

390          SXCALL $A35E          * __TGetWindowPos
391          move.l d0, -(sp)      * デフォルト位置に
392          move.l rgnHdl(a5), -(sp) * リージョンを移動させて
393          SXCALL $A161          * __GMMoveRgn
394          addq.l #8, sp
395
396          move.l d6, -(sp)      * プライオリティ
397          move.l rgnHdl(a5), -(sp) * リージョンの位置/形に
398          clr.l -(sp)          * サブウィンドウをオープン
399          SXCALL $A227          * __WSOpen
400          lea 12(sp), sp
401          move.l a0, (a4)+      * ポインタをリストに格納
402
403          subq.l #1, d6         * プライオリティ - 1
404          dbra d7, D61st0      * 繰り返し
405
406          rts
407
408 DrawGraphMain:
409
410          pea winPtr(a5)        * ウィンドウ内部を描画する
411          SXCALL $A131          * サブルーチン
412          addq.l #4, sp         * ウィンドウ内部を描画
413
414          move.l #$001c_0006, -(sp) * 描画位置の指定
415          SXCALL $A16E          * __GMMove
416          addq.l #4, sp
417          pea mainWinMsg(pc)    * 文字列を描画
418          SXCALL $A192          * __GMDrawStrZ
419          addq.l #4, sp
420
421          rts
422
423 DrawGraphSub:
424 * サブウィンドウ内部を描画するサブルーチン
425 * 引数
426 * D1 サブウィンドウレコードへのポインタ
427 * D2 サブウィンドウ番号
428 *
429          move.l d1, a2         * サブウィンドウレコードへの
430
431          pea (a2)              * ポインタ
432          SXCALL $A131          * 内部に描画
433          addq.l #4, sp         * __GMSetGraph
434
435          move.w #11, -(sp)     * フォアグラウンド
436          SXCALL $A147          * カラーは黒
437          addq.l #2, sp         * __GMForeColor
438          move.w #8, -(sp)     * 背景色は白
439          SXCALL $A148          * __GMBackColor
440          addq.l #2, sp
441          move.w #%1111, -(sp) * 4 ページ全部に描画
442          SXCALL $A149          * __GMAPage
443          addq.l #2, sp
444          move.w #$100, -(sp)  * 背景色で描画
445          SXCALL $A144          * __GMPenMode
446          addq.l #2, sp
447
448          move.l $4c(a2), -(sp) * インサイドリージョンを
449          move.l rgnHdl(a5), -(sp) * 作業用リージョンにコピー
450          SXCALL $A160          * __GMCopyRgn
451          addq.l #8, sp
452          move.l (a2), a0      * ビットマップレコードへの

```

452	move. l	2 (a0), -(sp)	* ポインタ
453	move. l	rgnHdl (a5), -(sp)	* グローバル座標系に
454	SXCALL	\$A162	* 作業用リージョンを変換
455	addq. l	#8, sp	* __GMSSlideRgn
456			
457	move. l	rgnHdl (a5), -(sp)	* 作業用リージョン内部を
458	SXCALL	\$A17B	* __GMFillRgn
459	addq. l	#4, sp	* で背景色で塗り潰す
460			
461	move. w	#\$00, -(sp)	* フォアグラウンドカラーで
			* 描画
			* __GMPenMode
462	SXCALL	\$A144	
463	addq. l	#2, sp	
464	move. l	rgnHdl (a5), -(sp)	* サブウィンドウの枠を描画
465	SXCALL	\$A17A	* __GMFrameRgn
466	addq. l	#4, sp	
467			
468	move. w	##0111, -(sp)	* アクセスページを
			* 0, 1, 2に
			* __GMAPage
469	SXCALL	\$A149	
470	addq. l	#2, sp	
471	move. w	#13, -(sp)	* フォアグラウンドカラーを
			* 赤に
			* __GMForeColor
472	SXCALL	\$A147	
473	addq. l	#2, sp	
474	pea	SWDragRect (pc)	* ドラッグリージョンを
475	SXCALL	\$A173	* __GMFillRect
476	addq. l	#4, sp	* で塗り潰す
477	move. w	#0, -(sp)	* 通常状態で
478	pea	SWCBoxRect (pc)	* クローズボタンの位置に
479	pea	SWCBoxPImg (pc)	* クローズボタンの
			* プロットイメージを
			* __GMPlotImg
480	SXCALL	\$A182	* で描画
481	lea	10 (sp), sp	
482			
483	move. w	#2, -(sp)	* 24ドットフォント
484	SXCALL	\$A18B	* __GMFontKind
485	addq. l	#2, sp	
486	move. w	#11, -(sp)	* フォアグラウンドカラーは黒
487	SXCALL	\$A147	* __GMForeColor
488	addq. l	#2, sp	
489	move. l	##0030_0002, -(sp)	* サブウィンドウ中央に
490	SXCALL	\$A16E	* __GMMove
491	addq. l	#4, sp	
492	pea	SWPri (pc)	* サブウィンドウの
			* プライオリティ (2ケタ)
			* __GMDrawStr2
493	SXCALL	\$A192	* で描画
494	addq. l	#4, sp	
495	move. w	#'0', d0	
496	add. w	d2, d0	
497	move. w	d0, -(sp)	* プライオリティの末尾を
498	SXCALL	\$A18F	* __GMDrawChar
499	addq. l	#2, sp	* で描画
500			
501	rts		
502			
503	_TINI:		* [終了処理]
504	cmp. l	##-2, d0	* 初期化を行なわなかった?
505	beq	_TINI3	* ならば_TINI3へ
506			
507	lea	sWinPtrList (a5), a4	* リストの先頭
508	move. l	##SWIN-1, d7	
509	_TINI0:		


```

510      move.l   (a4)+, d0      * クローズされている?
511      beq      _TINI1        *   ならば _TINI1へ
512      move.l   d0, -(sp)      * このサブウィンドウを廃棄
513      SXCALL   $A229          *   __WSDispose
514      addq.l    #4, sp
515  _TINI1:
516      dbra     d7, _TINI0      *   繰り返し
517
518      move.l    rgnHdl(a5), d0 * リージョンが
                                   *   作成されている?
                                   *   作成されていなければ
                                   *   廃棄
                                   *   __GMDisposeRgn
519      beq      _TINI2
520      move.l    d0, -(sp)
521      SXCALL   $A15B
522      addq.l    #4, sp
523  _TINI2:
524      pea      winPtr(a5)      * ウィンドウをクローズする
525      SXCALL   $A1FB          *   __WMClose
526      addq.l    #4, sp          *   WMDisposeでないことに注意
527  _TINI3:
528      moveq     #0, d0
529      rts
530
531      .even
532  winTitle:
533      dc.b      6, '付箋紙'    * ウィンドウタイトル
534  mainWinMsg:
535      dc.b      '主ウィンドウ', 0 * ウィンドウ内部に
                                   *   描画する文字列
536  SWPri:
537      dc.b      '10', 0        * プライオリティの
                                   *   上位2桁
538      .even
539  SWRect:
540      dc.w      0, 0, SWIN_X, SWIN_Y * サブウィンドウの
                                   *   大きさを意味する
                                   *   レクタングル
541  SWDragRect:
542      dc.w      0, 0, 16, 28    * サブウィンドウの
                                   *   ドラッグリージョンを
                                   *   意味するレクタングル
543  SWCBoxRect:
544      dc.w      2, 2, 12, 12  * サブウィンドウの
                                   *   クローズボタンの位置と
                                   *   大きさを意味する
                                   *   レクタングル
545  SWCBoxPImg:
546      dc.w      %00000000_00000000 * サブウィンドウの
                                   *   プロットイメージ
547      dc.w      %01111111_11000000
548      dc.w      %01111111_11000000
549      dc.w      %01111111_11000000
550      dc.w      %01111111_11000000
551      dc.w      %01111111_11000000
552      dc.w      %01111111_11000000
553      dc.w      %01111111_11000000
554      dc.w      %01111111_11000000
555      dc.w      %11111111_11000000
556
557      dc.w      %00000000_00000000
558      dc.w      %01000001_01000000
559      dc.w      %00100010_01000000
560      dc.w      %00010100_01000000
561      dc.w      %00001000_01000000
562      dc.w      %00010100_01000000
563      dc.w      %00100010_01000000
564      dc.w      %01000001_01000000
565      dc.w      %00000000_01000000

```

```

566          dc, w    %11111111_11000000
567
568          dc, w    %11111111_11000000
569          dc, w    %11111111_11000000
570          dc, w    %11111111_11000000
571          dc, w    %11111111_11000000
572          dc, w    %11111111_11000000
573          dc, w    %11111111_11000000
574          dc, w    %11111111_11000000
575          dc, w    %11111111_11000000
576          dc, w    %11111111_11000000
577          dc, w    %11111111_11000000
578
579 msgRec:
580          dc, w    13
581          dc, l    0
582          dc, l    0
583          dc, l    0
584          dc, w    $20
585 msgRec_Tskid:
586          ds, w    1
587
588          .end

```

■リスト6 SWSMPL用 makefile

```

# SWSMPL用 makefile

SWSMPL.X:      SKELTON.o SWSMPL.o
              lk -oSWSMPL SKELTON SWSMPL

SKELTON.o:     SKELTON.s WORK.INC
              as SKELTON

SWSMPL.o:      SWSMPL.s WORK.INC
              as SWSMPL

```

C 言語による プログラミング

前著『SX-WINDOW プログラミング』と本書では、SX アプリケーションのプログラミングをアセンブリ言語によって行う方法を述べてきました。アセンブリ言語による記述はプリミティブであるがゆえに、SX-WINDOW の仕組みが見えやすく、本質的な理解の助けになるという面があるものの、可読性に関してはいま一つです。一方、最近では C 言語によるプログラミングも主流となってきていますので、本書でも C 言語による開発について述べておきます。

4¹ C 言語とアセンブラの関係

SX アプリケーションを C 言語で作成する場合、通常の C コンパイラの動作環境に加えて、いくつかの準備が必要です。本書では、ユーザが C コンパイラを使って手軽に SX アプリケーションを作成できるよう、付録ディスクの「SXer Tool Box」に必要なファイルが収められています。ここでは、SX アプリケーションを作成するための C コンパイラのセットアップについて解説します。

プログラミングについて語るとき、それを記述する言語は本質的な問題ではありません。ただし、FORTH や LISP などのように、想定している CPU のアーキテクチャ自体がターゲットとしている MPU/CPU のアーキテクチャと大きく異なっているような場合は例外ですが。

ことに、私たちにもなじみ深い C 言語などは、「高級アセンブラ」という陰口を叩かれているほど「低級」な言語です。C プログラマの達人（少なくともパソコン上の処理系の達人）は、C のソースがどのようなコードに落とされるかを意識して書いているはずで、コンパイラを通すと思いの通りのコードが得られない場合は、インラインアセンブラで直接アセンブラのコードを書いてしまうこともあるでしょう。

一方、X68000 の MPU である MC68000 は、ほかの CPU と比較すると、アセンブラによる開発が非常に容易な部類に属します。簡潔で読みやすいニーモニック、強力なアドレッシングモード、豊富なレジスタなど、プログラマにとって有利な条件がいくつもあります。ちょっと気のきいたプリプロセッサでもあれば、C 言語に近い可読性を獲得することができ、さすがに大規模なプログラムや、複雑な計算などをとまなうものの記述には向きませんが、それ以外のプログラムならば、それほど面倒ではない、と個人的には考えています。

このように、少なくとも X68000 にかぎっていえば、C 言語とアセンブラの距離は非常に近く、本書で C 言語を持ち出すまでもない、と考えていました。

しかし、そうはいつでも、C 言語による開発は、アセンブリ言語にくらべれば楽です。C 言語はわかるけれども、アセンブラはいま一つ、という方も多いでしょうから、C 言語によるプログラミングについても触れないわけにはいかないでしょう。

また、SX-WINDOW のようなウィンドウシステムのアプリケーションは、オブジェクト指向プログラミングが有効であることが知られています。オブジェクト指向の言語処理系としては、C の拡張版である C++ がよく知られています³、すでに X68000 でも G++ (GNU C++) が動いていますし、XC をベースとした C++ 処理系が登場する可能性もあります。C++ で SX アプリケーションが開発できるようになるまで、その前段階として、C によるプログラミングを習得しておくことも、決して無駄ではないはずです。

1 開発に必要な環境

X68000で利用できる C 言語の処理系には現在いくつかの種類がありますが、本書では、もっとも一般的なシャープの C コンパイラ PRO-68K (以下, XC2) を利用する場合を例にとることにします。XC2 は、方言というほどの方言もない、比較的素直な ANSI 準拠コンパイラですから、ほかの処理系 (GNU C コンパイラ等) を利用する場合でも、手直しは最小限ですむと思われます。

XC2 で SX アプリケーションを開発するためには、XC2 が動作する環境が必要なことはいうまでもありませんが、このほかにもいくつかのファイルが必要です。以下に挙げるファイルは付録ディスクに収めてありますので、APPENDIX の「付録ディスク『SXer Tool Box』の利用」をよくお読みのうえ、ご利用ください。

●SXLIB.L

SX コールを利用するためのライブラリです。

このライブラリは、SX 開発キットに収められている純正ライブラリを、シャープ株式会社のご好意により、その中のごく一部の関数を除いて収録させていただきました。この中には、XC1 等で利用できる、A 形式のライブラリも収録されています。

このファイルは、ほかのライブラリと同様、環境変数 lib で指定されたディレクトリに置いてください。

●SXLIB.H

SXLIB.L の関数群のプロトタイプ宣言等を行うヘッダファイルです。

このファイルは、ほかのヘッダファイル同様、環境変数 include で指定されたディレクトリに置いてください。

●SXDEF.H

SXLIB.H の中でインクルードされるタイプの宣言等を行うヘッダファイルです。

このファイルは、ほかのヘッダファイル同様、環境変数 include で指定されたディレクトリに置いてください。

●MAIN_R.O

R タイプのモジュールを作成する場合のスタートアップです。C タイプのモジュールを作成する場合には、SXLIB.L 中のスタートアップが使用されるため、必要ありません。

このファイルは、環境変数 lib で指定されたディレクトリに置いてください。

すでに XC2 による開発環境が整っている方ならば、以上のファイルを、それぞれ指定のディ

レクトリに収めるだけで、SX アプリケーションの開発準備は完了です。

コンパイルに必要な RAM やディスクの容量は、SX アプリケーション以外を開発する場合とそれほど変わりません。幸い、XC2 はかろうじてフロッピーベースの、主記憶 2M バイトでも利用可能なので、投資は最小限ですみます（が、せめてハードディスクくらいは増設されることをおすすめします）。

4²

C 言語による開発の制限事項

SX-WINDOW の実行ファイルは、基本的には Human の実行ファイルと同形式です。しかし、そのプログラムの内容には SX アプリケーション独自の約束事が存在することはご存じのとおりです。たとえば、「テキストエリアとワークエリアの使い分け」であるとか、「リエントラントなコードの記述」といったことは、Human 上で実行するプログラムでは考慮する必要のなかった概念です。

XC2 などは、もともと Human 上で実行されるプログラムを生成することを目的として設計されている処理系ですから、こうした SX の約束事は当然ながら考慮されていません。そのため、XC2 等によって生成されるプログラムは、理想的なまでに SX アプリケーションらしい SX アプリケーションとはなりません。

XC2 で SX アプリケーションを作成する場合に特有な、少々厄介な問題について説明します。

(1) 原則として C 型のモジュールを生成する

XC2 は static な変数、グローバル変数をデータセクションに置きます。また、ライブラリ等が内部で静的な変数を用意している場合があるので、完全にリエントラントなプログラムを生成するのは難しくなっています。

このため、XC2 をベースとする開発キットは C 型のモジュールを作成することを前提としてつくられています。

R タイプのモジュールを作成する方法は用意されていますが、そこにはいくつかの制限事項があります。

O タイプの作成はできませんが、C タイプ用のスタートアップ (SXLIB.L 中の `_MAIN.O`) のモジュールヘッダを 'OBJC' から 'OBJO' に書き換えるだけなので難しいことはありません。O タイプのモジュールをつくる場合は、O タイプ用のスタートアップを作成してリンクするようにするとよいでしょう。

(2) (C の) ヒープの扱いが SX-SYSTEM となじまない

XC2 には `malloc ()` 等のヒープを扱う関数が用意されています (ここでいうヒープは、メモリマネージャのヒープゾーンとは異なりますので、注意してください)。これによるヒープの処理は、SX-SYSTEM とはあまりなじみがよくありません。

Human 上では、XC2 で作成したプログラムは、ヒープを扱う、扱わないに関わらず、起動直後に自分専用のヒープとして 64K バイトのメモリブロックを OS に請求します*1。この中から、`malloc ()` などで請求されただけメモリブロックを切り分けて、アプリケーションに与えます。この、ヒープゾーンとして確保された 64K バイトのメモリブロックは、これ以上のサイズに拡張されることはあっても、不必要になった分を OS に返却することはありません。

ません*2。

SX-SYSTEM 上でも、実現する方法こそ異なりますが、これと同じ手法がとられます。

C のヒープはワークエリア中に用意されます。デフォルトでは、ヒープとして 64K バイト、スタックとして 64K バイト、計 128K バイトが最低でも確保されます。何もしないプログラムでも、これは同じです。ありていにいえば、非常に無駄にメモリが消費されてしまうわけですが、これを理由に XC2 を責めるのは多少酷ではあります。

*1: コマンドラインから /HEAP: 128 などと指定することによってサイズを指定することは可能です。逆に、こうしたオプションを指定しないかぎり、つねに 64K バイトが自動的に確保されてしまう、ということでもあります。CLIB.L、あるいは SXLIB.L の中の `_MAIN.0` を修正することによって、デフォルトのサイズを変更することは可能ですが、あまり柔軟性のある方法とはいえません。

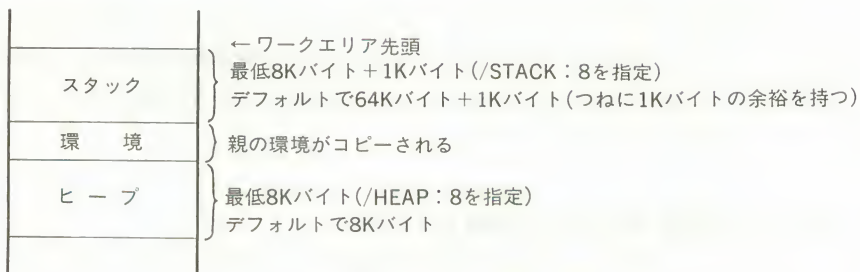
*2: Human では、`sbrk()` 関数などでヒープを拡張することが可能ですが、SX アプリケーションではこの関数は使用不可です。

(3) ワークエリアはスタック+ヒープとして使用、コモンエリアは使用不可

XC2 で使われる変数は、大きく static 変数と auto 変数の 2 種類に分けられます。このうち、static 変数 (XC2 では、グローバル変数も static として扱われます) はデータセクションに置かれ、ダイレクトアドレッシングでアクセスされます。auto 変数はスタックフレーム上に作成され、レジスタ A6 をベースとしてアクセスされます。

これらはコンパイラの仕様なので、SX アプリケーションを作成する場合でも同じです。一方、SX アプリケーションでは、ワークエリア中に変数を置くのがスマートな方法です。この折衷案として、XC2 で作成した SX アプリケーションは、ワークエリアを図 1 のように利用しています。

■図 1 ワークエリアの使用状況



グローバル変数をワークエリア中に作成できればリエントラントとなるのですが、これが R 型のモジュールを作成する場合の障害となっています。

また、コモンエリアは利用できません。

(4) 一部の DOS コール関数、IOCS 関数は使用不可

アセンブリ言語で作成した SX アプリケーションでも、次のような DOS コール、IOCS

コールは使用できません。

- ・VRAM (テキスト, グラフィックともに) を操作したり, 画面モードを変更したりする DOS/IOCS コール
- ・マウスやソフトキーボードを操作する DOS/IOCS コール
- ・プロセス管理に直接タッチする DOS コール

C 言語で作成した SX アプリケーションでも同様に, これら呼び出す関数は利用できません。

このほかに,

- ・シグナル関係
- ・ストリーム入出力関数の一部 (fcloseall () 等)
- ・メモリブロックを操作する関数の一部 (allmem (), sbrk () 等)
- ・標準入出力を利用する関数 (printf (), scanf () 等)
- ・BASIC ライブラリのほとんど

等も使用できないと考えたほうが無難です。

(5) R 型のモジュールを作成する場合の制限事項

C 型のモジュールが前提であるとはいえ, R 型のモジュールも制限付きながら作成可能です。この場合, 次のような制限が存在します。

- ① static 型変数, グローバル変数は使用できません。使用する場合は, 副作用をよく理解したうえで使用する必要があります。したがって, 変数はすべて auto 変数として作成してください。
- ② ヒープやファイルの管理は, static な変数によって管理されています。したがって, コードを共有するタスク間では, これらの管理は共通となります。混乱なく, これらを利用するには, かなり深くシステムを知る必要があるでしょう。
- ③ スタックチェックオプションは使用できません。各タスクにワークエリアの中にスタックが用意されるからです。

R 型のモジュールを作成する場合は, スタートアップとして SXLIB.L の中の `__MAIN.O` のかわりに `__MAINR.O` をリンクして使用します。こうした具体的なコンパイル/リンクの方法は, 次の項で述べることにします。

以上のように、C言語による開発にはいろいろと厄介なこともあります。C言語で開発を行う場合、あらかじめ、こうした事項を頭に入れておいてください。それによって得られるメリットを天秤にかけて、C言語で開発するか、アセンブリ言語にするか、適当な言語を選択する必要があるでしょう。

4³ SXLIB.L

SXLIB.L は、SX アプリケーションを作成するために必要なオブジェクトを収めたライブラリファイルです。XC2 で SX アプリケーションを作成する場合は、この中に収められている関数等を利用して SX-SYSTEM を利用することになります。

1 XC2 の出力する実行ファイルの構成

ここで少し、XC2 の出力するコードの内容について触れておきましょう。C のエキスパートの方は飛ばして下さってかまいません。

コンパイラの出力した実行ファイルには、プログラマが C 言語で記述したものをネイティブコードに変換したコードはもちろんですが、そのほかにもコンパイラが付加したコードが含まれています。

1 つは、ライブラリからリンクした関数のルーチンです。たとえば、プログラム中で `printf()` という関数を使用していれば、コンパイラは `printf()` を実現するためのコードを収めたオブジェクト、`PRINTF.O` を `CLIB.L` の中から探し出してリンクしてくれます。

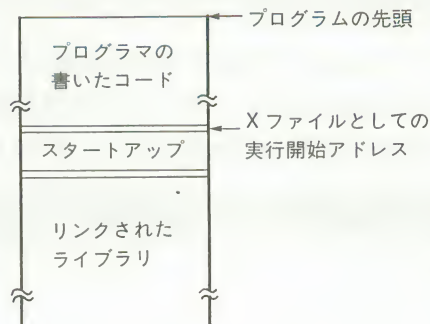
もう 1 つは、スタートアップと呼ばれる、プログラムを実行する準備を整えてくれるコードです。実行ファイルが起動されると、まず、このコードが実行され、準備が整ったところでプログラマの書いた `main()` に相当するコードの実行が始まります。Human 上で動作するプログラムの場合、`CLIB.L` の中に含まれる `__MAIN.O` というオブジェクトが、これにあたります。

Human のプログラムを作成する場合、これらのコードは `CLIB.L` というライブラリファイルに収められたものが使われます。`CLIB.L` は、XC2 で標準的に使われる関数のオブジェクトやスタートアップを収めたライブラリファイルなのです。ライブラリファイルとしては、`CLIB.L` のほかに `DOSLIB.L`、`IOCSLIB.L`、`BASLIB.L`、`FLOATFNC.L`、そして `FLOATEML.L` などが用意されていますが、それぞれファイル名が示しているようなコードを含んでいます。必要がある場合、これらをリンクするようコンパイラに指示して使用します。もちろん、ユーザがこれら以外のライブラリを用意して、リンクさせることも可能です。

以上のようなコードで構成される実行ファイルは、一般的に 254 ページの図 1 のように構成されています*1。

*1:あくまでも、非常に単純な C のソースに、`CLIB.L` のみ（あるいは、それに `FLOAT???.L` が加わる場合も含めて）を非常にシンプルな手順でリンクした場合を想定しています。

■図1 XC2 の生成した実行ファイルの構成の例



2 SXLIB.L のスタートアップ

CLIB.L が Human 上で動作するプログラムに基本的なコードを提供するのと同様に、SXLIB.L は SX アプリケーションのためのコードを提供するライブラリです。その中には、SX アプリケーション用のスタートアップ (SXLIB.L 中の `__MAIN.O`) や、SX コールを呼び出すための膨大な数の関数のオブジェクトが収められています。

SX アプリケーションを作成する場合、XC2 のコンパイルドライバ^{*2}に、C 言語によって記述された SX アプリケーションのソースをコンパイルした後、リンク時には SXLIB.L をリンクしろ、と指示します。たとえば、`foo.c` というソースに SXLIB.L をリンクする場合は、次のようにコンパイラを起動することになります。

```
A>CC foo.c SXLIB.L
```

CLIB.L はつねにリンクされることになっているので、とくに指定しなくても、(この場合) SXLIB.L の次にリンクされます。このとき、SXLIB.L の中には SX アプリケーション用のスタートアップ `__MAIN.O` が、CLIB.L の中には Human 上のプログラム用のスタートアップ `__MAIN.O` (同名です) が含まれているわけですが、先に SXLIB.L とのリンク作業が行われた場合、SXLIB.L の中にある `__MAIN.O` がスタートアップとしてリンクされることになります。

^{*2}:XC1 の場合、コンパイラはプリプロセッサ CPP.X、パーサ CC0.X、コードジェネレータ CC1.X、オブティマイザ CC2.X というぐあいに、各部が独立した実行ファイルとなっており、これらをコンパイルドライバ CC.X が呼び出してコンパイル作業を統括していました。XC2 になって、これらはすべて 1 つの実行ファイル CC.X にまとめられてしまい、独立した存在としてのコンパイルドライバはなくなってしまいました。ここでは、CC.X に含まれるコンパイルドライバとしての機能を指して、こう呼ぶことにします。

SXLIB.L 中のスタートアップは、起動されたプログラムが SX アプリケーションとして動作できるように準備を行います。その内容は、次のようなものです。

- ・モジュールヘッダの用意
- ・コマンドラインから起動された場合の、外部カーネルの呼び出し
- ・ワークエリア内の設定（スタック、ヒープの用意と、そのためのコマンドラインの解析）
- ・argc, argv の作成
- ・ストリーム入出力のための初期設定
- ・時間関数のための初期設定
- ・乱数の初期化

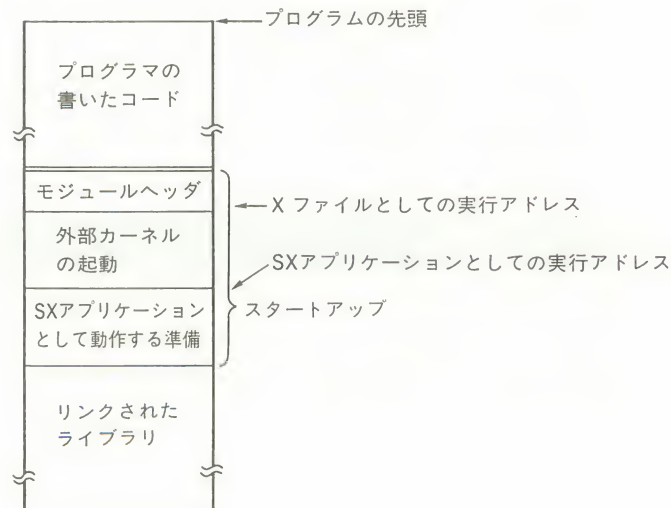
これらは Human 上のプログラムのための
スタートアップとほぼ同様

このような処理が行われた後、argc, argv をスタックに積んで、プログラマの書いた main () を呼び出すわけです。

これは、つまり、本書で示したアセンブラ版のスケルトン SKELTON.S の前半部分、74 行までに相当すると考えてよいでしょう。プログラマが C 言語で記述するのは、SKELTON.S の 74 行以降ということになります。

SXLIB.L をリンクした場合の実行ファイルは、図 2 のような構成となります。

■図 2 SXLIB.L をリンクした場合の実行ファイルの構成の例



main () の処理が終了すると、スタートアップに処理が戻ってきます。スタートアップは残りの処理、つまり \$A352 TSEExit を呼び出す処理を実行して、プログラムを終了させます。

SXLIB.H の中に含まれるスタートアップは C 型のモジュール用のスタートアップです。R 型のモジュールを作成したい場合、このスタートアップのかわりに別ファイルとして提供されている `__MAINR.O` をリンクしなければなりません。そのためには、たとえば R 型のモジュールとして記述された `bar.c` というソースがあった場合、次のようにコンパイラを起動することになります。

```
A>CC bar.c __MAINR.O SXLIB.L
```

3 SXLIB.L によってサポートされる関数

DOS コールを呼び出す関数が DOSLIB.L の中に収められています。たとえば、MPU の特権状態を切り替えるための DOS コール、`$F20 SUPER` を呼び出したい場合、C のソース中で関数 `SUPER ()` を呼び出せば、コンパイル/リンク時に、DOSLIB.L の中の `DOS20.O` がリンクされることになります。

これと同様に、SX コール 1 つ 1 つに、それに対応する C の関数が用意されています。

たとえば、`$A357 TSEventAvail` に対応する C の関数としては、`int TSEventAvail (int, tsevent*)` が用意されています。これは、SXLIB.L の中では `A357.O` というオブジェクトとして登録されています。

このように、SX コールを呼び出す関数には、おおむね SX コール名そのままの関数名が与えられています。付録ディスク「SXer Tool Box」に収められている SXLIB.L に含まれる関数のリファレンスについては、やはり付録ディスクのディレクトリ `¥C 開発キット ¥DOC` 中の `SXLIB.DOC` の一覧表を参照してください。ここには、各関数の引数と返り値、そしてかんたんな注意事項などが記されています。それぞれの機能については同名の SX コールのリファレンスを参照してください。

4 SXLIB.H

実際に SXLIB.L を利用する場合、ソースの先頭で SXLIB.H を `include` する必要があります。SXLIB.H では、SXLIB.L に含まれる関数のプロトタイプ宣言等を行うほか、構造体やシンボル類を定義している `SXDEF.H` の `include` 等を行っています。

4 C 言語版スケルトン

C 言語で記述するからといって、SX アプリケーションのスタイルが変わるわけではありません。アセンブリ言語で記述していたときと同様、初期化して、イベントを待って、各イベントの処理ルーチンを実行して、そして指示があったら終了処理を行う、というぐあいに、おなじみの仕事をこなすだけの話です。

なにはともあれ、C 言語版のスケルトンを見てください（リスト 1）。

■リスト 1 C 版スケルトン

```

1  /*                                     */
2  /*             SX-WINDOW             */
3  /*             C 版スケルトン         */
4  /*                                     */
5
6  #include <STDIO.H>
7  #include <STDLIB.H>
8  #include <SXLIB.H>
9
10 #define WINOPT 0b0000                /* ウィンドウオプション */
11 #define WIN_X 256                    /* ウィンドウ初期 x      */
12 #define WIN_Y 128                   /* ウィンドウ初期 y      */
13
14 int      IdleEvent( void );
15 int      LeftDownEvent( void );
16 int      LeftUpEvent( void );
17 int      RightDownEvent( void );
18 int      RightUpEvent( void );
19 int      KeyDownEvent( void );
20 int      KeyUpEvent( void );
21 int      UpdateEvent( void );
22 int      ActivateEvent( void );
23 int      System12Event( void );
24 int      Init( void );
25 int      DrawGraph1st( void );
26 void     DrawGraph( void );
27 void     Tini( int );
28
29 /*      グローバル変数      */
30
31 rect     winRect = { 0, 0, WIN_X, WIN_Y }; /* ウィンドウレクタングル */
32 int      paramFlg = 0;                  /* コマンドラインの解析結果 */
33 tsevent   eventRec;                     /* イベントレコード      */
34 int      eventMask = 0xffff;            /* イベントマスク        */
35 int      taskId;                         /* タスク ID             */
36 window    *winPtr = 0;                  /* ウィンドウレコードへのポインタ */
37 int      winActive = 0;                  /* アクティブフラグ      */
38
39 /*      メイン      */
40 int      main()
41 {
42     int      status;
43
44     status = Init();                    /* アプリケーションの初期化 */
45     while ( status >= 0 ) {              /* 返り値が正の数である間ループ */
46         TSEventAvail( eventMask, &eventRec );
47         switch ( eventRec.what ) {
48             case E_IDLE:                 /* アイドルイベント      */

```

```

49         status = IdleEvent();
50         break;
51     case E_MSLDOWN:      /* レフトダウンイベント */
52         status = LeftDownEvent();
53         break;
54     case E_MSLUP:        /* レフトアップイベント */
55         status = LeftUpEvent();
56         break;
57     case E_MSRLDOWN:     /* ライトダウンイベント */
58         status = RightDownEvent();
59         break;
60     case E_MSRLUP:       /* ライトアップイベント */
61         status = RightUpEvent();
62         break;
63     case E_KEYDOWN:      /* キーダウンイベント */
64         status = KeyDownEvent();
65         break;
66     case E_KEYUP:        /* キーアップイベント */
67         status = KeyUpEvent();
68         break;
69     case E_UPDATE:       /* アップデートイベント */
70         status = UpdateEvent();
71         break;
72     case E_ACTIVATE:     /* アクティベートイベント */
73         status = ActivateEvent();
74         break;
75     case E_SYSTEM1:      /* システムイベント1 */
76     case E_SYSTEM2:      /* システムイベント2 */
77         status = System12Event();
78         break;
79     }
80 }
81 Tini( status);          /* アプリケーションの終了処理 */
82 }
83
84 /* アイドルイベント */
85 int IdleEvent( void)
86 {
87     return 0;
88 }
89
90 /* レフトダウンイベント */
91 int LeftDownEvent( void)
92 {
93     int part;
94     /* 自分のウィンドウ上で発生した? */
95     if ( winPtr == ( window *) ( eventRec.whom) ) {
96         if ( winActive == 0) /* インアクティブならば */
97             WMSelect( winPtr); /* アクティブに */
98         else {
99             part = SXCallWindM( winPtr, &eventRec);
100             switch ( part) {
101                 case W_INCLOSE: /* クローズボタンが押された */
102                     return -1; /* 終了へ */
103                     break;
104                 case W_ININSIDE: /* ウィンドウコンテンツ */
105                     /* なんにもしない */
106                     break;
107             }
108         }
109     }
110     return 0;
111 }
112

```



```

113 /*      レフトアップイベント      */
114 int      LeftUpEvent( void)
115 {
116         return 0;
117 }
118
119 /*      ライトダウンイベント      */
120 int      RightDownEvent( void)
121 {
122         return 0;
123 }
124
125 /*      ライトアップイベント      */
126 int      RightUpEvent( void)
127 {
128         return 0;
129 }
130
131 /*      キーダウンイベント      */
132 int      KeyDownEvent( void)
133 {
134         return 0;
135 }
136
137 /*      キーアップイベント      */
138 int      KeyUpEvent( void)
139 {
140         return 0;
141 }
142
143 /*      アップデートイベント      */
144 int      UpdateEvent( void)
145 {
146         WMUpdate( winPtr);
147         DrawGraph();
148         WMUpdtOver( winPtr);
149         return 0;
150 }
151
152 /*      アクティベートイベント      */
153 int      ActivateEvent( void)
154 {
155         /* 自分のウィンドウが
156            アクティブに? */
157         if ( winPtr == ( window *) ( eventRec.whom) )
158             winActive = 1;
159         /* アクティブフラグをセット */
160         else
161             winActive = 0;
162         /* アクティブフラグをリセット */
163         return 0;
164 }
165
166 /*      システムイベント 1, 2      */
167 int      System12Event( void)
168 {
169         switch ( eventRec.what2) {
170             case ENDTSK:
171                 /* タスクの終了 */
172                 /* 全ウィンドウのクローズ */
173                 /* 終了へ */
174                 return -1;
175             case WINDOWSELECT:
176                 /* ウィンドウのセレクト */
177                 /* ウィンドウをセレクトする */
178                 WMSelect( winPtr);
179             }
180         return 0;
181 }

```

```

176 /* アプリケーションの初期化を行なう */
177 int Init( void)
178 {
179     task    tbuff;          /* タスク管理テーブルを
                               コピーしてくる */
180
181     TSGetTdb( &tbuff, -1);  /* タスク管理テーブルを
                               コピーする */
182     paramFlg = TSTakeParam( &tbuff.command, &winRect, 0, 0, 0, 0);
183     if ( ( paramFlg & 1) == 0) { /* -Wオプションが
                                   指定されていない場合 */
184         *(long *)(&winRect.left) = TSGetWindowPos();
185         winRect.right = winRect.left + WIN_X;
186         winRect.bottom = winRect.top + WIN_Y;
187     }
188     taskId = TSGetID();      /* タスクIDを得る */
189                               /* ウィンドウを開く */
190     winPtr = ( window *)WMOpen( ( window *) ( 0),
                                   /* ヒープ上に作成 */
                                   &winRect,
                                   /* ウィンドウレクタンクル */
                                   ( LASCII *) ( "¥010NOTITLE"),
                                   /* ウィンドウタイトル */
                                   -1, /* 可視 */
                                   ( 0x20 << 4) + WINOPT,
                                   /* 標準ウィンドウ */
                                   ( window *) ( -1),
                                   /* もっとも手前に */
                                   -1, /* クローズボックスあり */
                                   ( long) ( taskId) );
                                   /* タスクID */
191
192     if ( winPtr == 0)        /* エラー? */
193         return -1;          /* ならば終了へ */
194
195     return ( DrawGraphlst()); /* ウィンドウの内部を描画する */
196 }
197
198 /* ウィンドウ内部の描画の
199 準備を行なう関数
200 int DrawGraphlst( void)
201 {
202     GMSetGraph( ( graph *) ( winPtr)); /* グラフポートをセット */
203     return 0;                          /* なにもしない */
204 }
205
206 /* ウィンドウ内部を描画する関数
207 void DrawGraph( void)
208 {
209     GMSetGraph( ( graph *) ( winPtr)); /* グラフポートをセット */
210     /* なにもしない */
211 }
212
213 /* 終了処理
214 void Tini(int status)
215 {
216     if ( winPtr != 0)        /* ウィンドウが開かれていたら */
217         WMDispose( winPtr); /* 廃棄する */
218     exit( 0);
219 }

```

SKELTON.C は、アセンブラ版スケルトンである SKELTON.S と BODY.S の内容を、そのまま C に置き換えたものです。機能的にはほとんど同じで、ウィンドウを 1 つ開いて、それがドラッグ、クローズできるというだけの代物です。ですが、必要な骨格は備えていますから、肉付けすることによって実用的なアプリケーションを作成することが可能です。

このスケルトンをコンパイルしてみましょう。

A>CC SKELTON.C SXLIB.L

ヘッダファイルが大きいので、プログラムのサイズのわりには時間がかかると思います。

付録ディスクに収めたソースをコンパイルしてみてエラーが発生するようでしたら、ライブラリやヘッダファイルのインストールに問題があると思われます。もう一度チェックしてみてください。

コンパイルの結果、正常に実行ファイル SKELTON.X が作成されたら、コマンドラインから、あるいは SX シェル上から実行してみてください。その際、「プロセス情報」を走らせておくと、メモリの消費ぐあいが確認できます。たったこれだけのプログラムですが、実行ファイルのサイズは 6K バイト強、実行時にはトータルで 140K バイトほどのメモリを消費してしまいます。

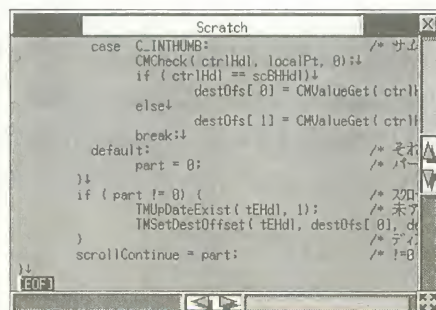
4^{…5} サンプルプログラム

C言語版のスケルトンをベースとして作成したサンプルプログラムを示します。サンプルプログラム CSAMPLE は、テキストエディットを利用した、かんたんなエディタを実現するプログラムです。

1 プログラムの仕様

CSAMPLE.EX を起動すると、スクロールバー付きの標準ウィンドウ（ウィンドウタイトル 'Scratch'）が表示され、内部でカーソルが点滅を始めます（図 1）。

■図 1 CSAMPLE の実行例



カーソルが点滅しているのは、ウィンドウがアクティブな場合です。この状態でキーボードから文字を入力することができます。ウィンドウがインアクティブの場合は、カーソルは点滅せず、文字の入力もできません。

マウスの左ボタンによって、ウィンドウのドラッグ、サイズ変更、ズームイン/アウト等を行うことができます。また、テキスト内部でクリック/ドラッグすることにより、カーソルの移動/セレクト範囲が可能です。

マウスの右ボタンでは、テキストエディットのポップアップメニューが表示され、デスクトップスクラップとの間でカット&ペーストが可能です。キーボードによるショートカットはサポートしていません。

2 プログラムの説明

265 ページからのプログラムは、テキストエディットを利用したエディタの非常にプリミティブなモデルです。文書を編集するための必要最低限の機能は有しています。

65 行目からは main 関数です。行っていることはほとんどスケルトンと同じですが、使用していないイベントへの分岐は省略しています。

101 行目からはアイドルイベントの処理を行う関数です。ウィンドウがインアクティブの

場合は何もしません。アクティブの場合は、まずスクロールバーを操作中かどうかを調べ、左ボタンが押され続けているのならば、スクロール処理を続けます。左ボタンが離されていれば、スクロールバーの処理を終了します。

スクロールバーの処理を行っていない場合、定期的にスクロールバーの再描画を行います。本来ならば、スクロールバーの値が前回調べたときから変化した場合にのみ再描画を行えばよいのですが、ここではその判断を省略しています。この処理を行わない場合は、TMEventW () を利用してカーソルの点滅を行います。

122 行目からはレフトダウンイベントの処理関数です。SXCaliWindM () を利用して分岐を行うのはスケルトンと同様で、サイズボタン関係、スクロールバー関係、テキストのセレクト等に関する処理が追加されています。テキストのセレクト処理は、144 行の TMEventW () を呼び出す 1 行だけで、後はテキストマネージャが適当に判断して処理を行ってくれます。

157 行目からのライトダウンイベントの処理関数、166 行目のキーダウンイベント処理関数では、実質的に TMEventW () を呼び出しているだけです。ここでもやはり、テキストマネージャの判断によって、これだけでカット&ペーストの処理が行われているわけです。

174 行からのアップデートイベントで注意していただきたいのは、アップデートを行う前にカーソルを消している点です (177~178 行目)。カーソルを消してから書き換えを行わないと、それ以降のカーソルの描画が正常に行われない場合があるので、必ずこの処理を行うようにしてください*1。

*1: 前著「SX-WINDOW～」でもテキストエディットを使用したサンプルプログラムを掲載しましたが、ここではカーソルを消していないため、カーソルの描画がおかしくなる場合があります。「SX-WINDOW～」のプログラムをアセンブル/実行して、ほかのウィンドウをカーソルの上に重ねたり除いたりすることで、この問題を確認することができます。

188 行目からのアクティベートイベントの処理関数では、ウィンドウがインアクティブになった場合、カーソルを点燈させる処理を行っています。同時にアクティブフラグを OFF にしていますので、アイドルイベントによるカーソルの点滅処理は行われず、結果的にカーソルは点燈したままの状態となります。

214 行目からの Init () は、アプリケーションの初期化を行う関数です。スケルトンとあまり違いはありませんが、ウィンドウオープン後、サイズボタンを使用するためにレコード内のフラグを操作している点で異なっています。

244 行目からの DrawGraph1st () はウィンドウ内部の初期描画、あるいはそのための諸設定を行う関数です。ここで行っているのは、おもにテキストエディットとスクロールバーのオープン、そして、それらの最初の描画です。テキストエディットのオープン時にキャッシュを ON にしています (253 行目)。これは SX1.10 からの機能ですが、キャッシュを ON にしなかった場合とのスピードの差は歴然としています。この行 (と次の行) を注釈にして確かめてみることをおすすめしておきます。

273 行目からの DrawGraph () は、このプログラムではアップデート時の描画を行うためだけの存在となってしまいました。行っていることは単純で、TMUpdate3 () を呼んでテキストエディット部分を、CMDraw () でスクロールバーをそれぞれ再描画して、その後サイズボタンを描画しています。

282 行目からは終了処理を行う関数です。テキストエディットとスクロールバー、ウィンドウの廃棄を行って exit () しています。

これ以降にはウィンドウ内部の座標の処理等を行う関数が置かれています。計算が多く、一見すると、かなりややこしく思えるかもしれませんが、どちらかという、比較的単純な計算を数多くこなすという、退屈な処理といえるかもしれません。とくに、ビューレクタングル等の大きさをウィンドウコンテンツの大きさにあわせて設定する SetTextRect () (294 行目～)、スクロールバーの大きさをウィンドウにあわせて計算し直す SetScBRect () (304 行目～)、現在のビューレクタングルとテキスト全体の位置関係を計算し、スクロールバーの値として設定する SetScBValue (317 行目～) などは、非常によく使われる関数ですから、毎回書きおろすよりはライブラリ的に使い回したほうが楽です。

341 行目以降の UpdateText (), ScrollText () の2つの関数は、その前の計算主体の関数を利用して目に見える処理を行います。前者は、ウィンドウの大きさをもとに計算した各種の値からテキストエディットやスクロールバーの大きさなどを求め、実際に再描画する関数。後者は、マウスの押されているポイントから、スクロールバーの処理を行って再描画する (UpdateText () 利用) 関数です。

このプログラムには、じつは問題があります。スクロールバーの取りうる値は -32768～32767、実際には最小値 0 として使われることが多いので、0～32767 というところです。この値 1 に対して、画面上の 1 ドットが相当しています。では、縦方向に 32767 ドット以上 (6×12 ドットフォントならば約 2730 行) のサイズをもつテキストを扱いたい場合はどうしたらよいのでしょうか？*2 これは各自で考えてみてください。

*2: もっとも、このプログラムではテキストエディットを開く際に最大バイト数を 64K バイトに限定しているので、2730 行ものテキストを扱えるかどうかは怪しいところです。

3

プログラムリスト

リスト 1 に CSAMPLE.C を示します。これをコンパイルする場合は、

```
A>CC CSAMPLE.C SXLIB.L
```

のように行います。

■リスト1 CSAMPLE.C

```

1  /*                                     */
2  /*   SX-WINDOW                         */
3  /*   CSAMPLE.C                       */
4  /*                                     */
5  /*   C言語によるサンプル             */
6  /*                                     */
7
8  #include <STDIO.H>
9  #include <STDLIB.H>
10 #include <SXLIB.H>
11
12 #define WINOPT          WC_SCROLL | WC_GBOX /* ウィンドウオプション */
13 #define WIN_X           256                /* ウィンドウ初期x      */
14 #define WIN_Y           256                /* ウィンドウ初期y      */
15 #define FONT_SIZE       6                 /* フォントのx方向のサイズ */
16 #define LINE_MAX        128               /* 1行に入る文字数      */
17 #define SCROLL_INTERVAL 20                /* スクロールを再描画する間隔 */
18 #define CACHE_SIZE      4096              /* キャッシュサイズ     */
19
20 typedef struct {
21     short min;
22     short max;
23     short value;
24 } scbVal; /* スクロール値を表現する型 */
25
26 int IdleEvent( void );
27 int LeftDownEvent( void );
28 int LeftUpEvent( void );
29 int RightDownEvent( void );
30 int RightUpEvent( void );
31 int KeyDownEvent( void );
32 int KeyUpEvent( void );
33 int UpdateEvent( void );
34 int ActivateEvent( void );
35 int System12Event( void );
36 int Init( void );
37 int DrawGraph1st( void );
38 void DrawGraph( void );
39 void Tini( int );
40 void SetTextRect( void );
41 void SetScBRect( void );
42 void SetScBValue( void );
43 void UpdateText( int );
44 void ScrollText( point_t, unsigned long );
45
46 /*   グローバル変数                     */
47
48 rect winRect = { 0, 0, WIN_X, WIN_Y }; /* ウィンドウレクタングル */
49 int paramFlg = 0; /* コマンドラインの解析結果 */
50 tsevent eventRec; /* イベントレコード */
51 int eventMask = 0xffff; /* イベントマスク */
52 int taskId; /* タスクID */
53 window twinPtr = 0; /* ウィンドウレコードへのポインタ */
54 int winActive = 0; /* アクティブフラグ */
55 tEdit tEdHdl = 0; /* テキストエディットレコードへのハンドル */
56 rect viewRect, destRect; /* テキストエディットのレクタングル */
57 control tscBHdl = 0; /* スクロールバー横へのハンドル */
58 control tscBVHdl = 0; /* スクロールバー縦へのハンドル */
59 rect scBHRect, scBVRect; /* スクロールバー縦横のレクタングル */
60 scbVal scBHVal, scBVVal; /* スクロールバー縦横の値 */
61 int scrollContinue = 0; /* スクロール継続中フラグ */
62 int lastTime = 0; /* スクロールバー最終再描画時刻 */
63
64 /*   メイン                             */
65 int main()
66 {
67     int status;
68
69     status = Init(); /* アプリケーションの初期化 */
70     while ( status >= 0 ) { /* 返り値が正の数である間ループ */
71         TSEventAvail( eventMask, &eventRec );
72         switch ( eventRec.what ) {
73             case E_IDLE: /* アイドルイベント */
74                 status = IdleEvent();
75                 break;
76             case E_MSLEDOWN: /* レフトダウンイベント */
77                 status = LeftDownEvent();
78                 break;
79             case E_MSRODOWN: /* ライトダウンイベント */
80                 status = RightDownEvent();
81                 break;
82             case E_KEYDOWN: /* キーダウンイベント */
83                 status = KeyDownEvent();
84                 break;
85             case E_UPDATE: /* アップデートイベント */
86                 status = UpdateEvent();
87                 break;

```

```

88         case E_ACTIVATE:      /* アクティベートイベント */
89             status = ActivateEvent();
90             break;
91         case E_SYSTEM1:       /* システムイベント1 */
92         case E_SYSTEM2:       /* システムイベント2 */
93             status = System12Event();
94             break;
95         /* その他のイベントは省略 */
96     }
97     Tini( status);           /* アプリケーションの終了処理 */
98 }
99
100 /* アイドルイベント */
101 int IdleEvent( void)
102 {
103     if ( winActive ) {      /* ウィンドウはアクティブ? */
104         GMSetGraph( ( graph *) ( winPtr));
105         if ( scrollContinue) /* スクロール-操作中? */
106             if ( ! EMLBttn()) /* 左ボタンは押されている? */
107                 if ( winPtr == ( window *) ( eventRec.whom))
108                     ScrollText( ( point_t) ( GMGlobalToLocal( ( point_t) ( &eventRec.whom2))),
109                                 ( unsigned long *) ( &eventRec.what2));
110     } else
111         scrollContinue = 0; /* スクロール中止 */
112     else if ( ( lastTime + SCROLLINTERVAL) < eventRec.when) {
113         UpdateText( 0); /* 定期的にスクロール-を再描画 */
114         lastTime = eventRec.when;
115     } else /* キャレットをブリンク */
116         TMEventW( tEHdl, ( event *) ( &eventRec));
117 }
118 return 0;
119 }
120
121 /* レフトダウンイベント */
122 int LeftDownEvent( void)
123 {
124     int part;
125     /* 自分のウィンドウ上で発生した? */
126     if ( winPtr == ( window *) ( eventRec.whom)) {
127         if ( winActive == 0) /* インアクティブならば */
128             WMSelect( winPtr); /* アクティブに */
129     } else {
130         part = SXCallWindM( winPtr, &eventRec);
131         switch ( part) {
132             case W_INCLOSE: /* クローズボタンが押された */
133                 return -1; /* 終了へ */
134             case W_INGROW: /* サイズボタンが押された */
135             case W_INZMOUT: /* ズームアウトした */
136             case W_INZMIN: /* ズームインした */
137                 UpdateText( 1); /* テキストとスクロール-を書き直す */
138                 break;
139             case W_ININSIDE: /* ウィンドウコンテンツ */
140                 if ( GMPTInRect( &viewRect,
141                                 GMGlobalToLocal( ( point_t) ( &eventRec.whom2))))
142                     /* ビューレクタングル中? */
143                     TMEventW( tEHdl, ( event *) ( &eventRec));
144                     /* セレクト処理等 */
145                 else ScrollText( ( point_t) ( GMGlobalToLocal( ( point_t) ( &eventRec.whom2))),
146                                 ( unsigned long *) ( &eventRec.what2));
147                 /* スクロール-の処理 */
148                 break;
149         }
150     }
151 }
152 return 0;
153 }
154
155 /* ライトダウンイベント */
156 int RightDownEvent( void)
157 {
158     /* 自分のウィンドウ上で発生した? */
159     if ( winPtr == ( window *) ( eventRec.whom))
160         TMEventW( tEHdl, ( event *) ( &eventRec));
161     /* メニューによるカット&ペースト処理 */
162     return 0;
163 }
164
165 /* キーダウンイベント */
166 int KeyDownEvent( void)
167 {
168     TMEventW( tEHdl, ( event *) ( &eventRec));
169     /* 文字入力 */
170     return 0;
171 }
172
173 /* アップデートイベント */
174 int UpdateEvent( void)
175 {
176     /* 自分のウィンドウ上で発生した? */
177     if ( winPtr == ( window *) ( eventRec.whom)) {
178         GMSetGraph( ( graph *) ( winPtr));

```



```

178         TMCaret( tEHdl, 0);          /* キャレットを消す */
179
180         WMUpdate( winPtr);            /* アップデート開始 */
181         DrawGraph();                  /* ウィンドウ内部を描画 */
182         WMUpdtOver( winPtr);          /* アップデート終了 */
183     }
184     return 0;
185 }
186
187 /* アクティブイベント */
188 int ActivateEvent( void)             /* */
189 {
190     /* 自分のウィンドウがアクティブに? */
191     if ( winPtr == ( window *) ( eventRec.whoml) )
192     {
193         winActive = 1;                /* アクティブフラグをセット */
194     }
195     else {
196         winActive = 0;                /* アクティブフラグをリセット */
197         TMCaret( tEHdl, 1);          /* キャレットを点燈する */
198     }
199     return 0;
200 }
201
202 /* システムイベント 1, 2 */
203 int SystemI2Event( void)             /* */
204 {
205     switch ( eventRec.what2) {
206     case ENOTSK:                      /* タスクの終了 */
207     case CLOSEALL:                    /* 全ウィンドウのクローズ */
208     case WM_CLOSE:                    /* 終了へ */
209         return -1;
210     case WINDOWSELECT:                /* ウィンドウのセレクト */
211     case WM_SELECT:                    /* ウィンドウをセレクトする */
212         WMSelect( winPtr);
213     }
214     return 0;
215 }
216
217 /* アプリケーションの初期化を行なう */
218 int Init( void)                      /* */
219 {
220     task tbuff;                      /* タスク管理テーブルをコピーしてくる */
221
222     TSGetFdb( &tbuff, -1);          /* タスク管理テーブルをコピーする */
223     paramFlg = TSfakeParam( &tbuff, command, &winRect, 0, 0, 0, 0);
224     if ( ( paramFlg & 1) == 0) {      /* -Wオプションが指定されていない場合 */
225         ( long *) (&winRect, left) = TSGetWindowPos();
226         winRect.right = winRect.left + WIN_X;
227         winRect.bottom = winRect.top + WIN_Y;
228     }
229     taskid = TSGetID();              /* タスクIDを得る */
230     /* ウィンドウを開く */
231     winPtr = ( window *) WMOpen( ( window *) ( 0), /* ヒープ上に作成 */
232     &winRect, /* ウィンドウレクタングル */
233     ( LASCII *) ( "¥007Scratch"), /* ウィンドウタイトル */
234     -1, /* 可視 */
235     ( 0x20 << 4) + WINOPT, /* 標準ウィンドウ */
236     ( window *) ( -1), /* もっとも手前に */
237     -1, /* クローズボックスあり */
238     ( long ) ( taskid)); /* タスクID */
239     if ( winPtr == 0)                /* エラー? */
240     {
241         return -1;                   /* ならば終了へ */
242     }
243     winPtr->wOption |= WC_GBOXON;    /* サイズボタン使用 */
244     return ( DrawGraphIst());         /* ウィンドウの内部を描画する */
245 }
246
247 /* ウィンドウ内部の描画の準備を行なう関数 */
248 int DrawGraphIst( void)             /* */
249 {
250     GMSetGraph( ( graph *) ( winPtr)); /* グラフポートをセット */
251     SetTextRect();                  /* テキストのウツツを決定 */
252     GMFontMode( 0);
253     if ( TMOpen( "65536, &destRect, 0, 12, &tEHdl) != 0)
254     {
255         return -1;
256     }
257     if ( ( tEHdl).drawMode == 0b000011; /* 改行コードとEOFを表示 */
258     if ( TMCacheON( tEHdl, CACHESIZE) < 0) /* キャッシュオン */
259     {
260         return -1;
261     }
262     TMSetRect( tEHdl, &destRect, &viewRect); /* テキスト/ビューレクタングルを設定 */
263     SetScBRect();                  /* x70-80のウツツを計算 */
264     SetScBValue();                 /* x70-80の最大/最小/現在値を計算 */
265     if ( ( scBHHdl = CMOpen( winPtr, &scBRect, ( LASCII *) ( " ", -1,
266     scBHVAl.value, scBHVAl.min, scBHVAl.max,
267     CI_SCLBRWH << 4, 0)) == 0)
268     {
269         /* x70-80 (横) をオープン */
270     }
271     return -1;
272     if ( ( scBVHdl = CMOpen( winPtr, &scBRect, ( LASCII *) ( " ", -1,
273     scBVVAl.value, scBVVAl.min, scBVVAl.max,
274     CI_SCLBRWH << 4, 0)) == 0)
275     {
276         /* x70-80 (縦) をオープン */
277     }
278     return -1;
279 }

```

```

268 UpdateText ( 1 );          /* いろいろ描画 */
269 return 0;
270 }
271
272 /* ウィンドウ内部を描画する関数 */
273 void DrawGraph( void)
274 {
275     GMSetGraph( { graph } ( winPtr) ); /* グラフポートをセット */
276     TMUpdate3( tEHdl, &viewRect); /* テキストをアップデート */
277     CMDraw( winPtr); /* スクロールバーを描画 */
278     WMDrawGBox( winPtr); /* サイズボタンを描画 */
279 }
280
281 /* 終了処理 */
282 void Tini( int status)
283 {
284     if ( tEHdl != 0) /* テキストエディットが開かれていたら */
285         TMDispose( tEHdl); /* 廃棄する */
286     if ( winPtr != 0) { /* ウィンドウが開かれていたら */
287         CMKill( winPtr); /* コントロール類を廃棄 */
288         WMDispose( winPtr); /* ウィンドウを廃棄 */
289     }
290     exit( 0 );
291 }
292
293 /* テキストエディットのレクタングルを計算する関数 */
294 void SetTextRect( void)
295 {
296     destRect.left = viewRect.left = winPtr->wGraph.grRect.left + FONTSIZE;
297     destRect.top = viewRect.top = winPtr->wGraph.grRect.top;
298     destRect.right = LINEMAX + FONTSIZE;
299     viewRect.right = winPtr->wGraph.grRect.right - 18;
300     destRect.bottom = viewRect.bottom = winPtr->wGraph.grRect.bottom - 18;
301 }
302
303 /* スクロールバーのレクタングルを計算する関数 */
304 void SetScBRect( void)
305 {
306     scBHRect.left = winPtr->wGraph.grRect.left;
307     scBHRect.top = winPtr->wGraph.grRect.bottom - 13;
308     scBHRect.right = winPtr->wGraph.grRect.right - 18;
309     scBHRect.bottom = winPtr->wGraph.grRect.bottom;
310     scBVRect.left = winPtr->wGraph.grRect.right - 18;
311     scBVRect.top = winPtr->wGraph.grRect.top;
312     scBVRect.right = winPtr->wGraph.grRect.right;
313     scBVRect.bottom = winPtr->wGraph.grRect.bottom - 18;
314 }
315
316 /* スクロールバーの値を計算する関数 */
317 void SetScBValue( void)
318 {
319     int xdots, ydots;
320     int i;
321
322     xdots = ( tEHdl).dest.right - ( tEHdl).dest.left;
323     if ( xdots < ( i = ( tEHdl).view.right - ( tEHdl).offsetH) )
324         xdots = i;
325     ydots = ( tEHdl).nLines * ( tEHdl).lineHeight;
326     if ( ydots < ( i = ( tEHdl).view.bottom - ( tEHdl).offsetV) )
327         ydots = i;
328
329     scBHVVal.min = 0;
330     scBHVVal.max = xdots - ( ( tEHdl).view.right - ( tEHdl).view.left);
331     scBHVVal.value = ( tEHdl).dest.left - ( tEHdl).offsetH;
332     scBHVVal.min = 0;
333     scBHVVal.max = ydots - ( ( tEHdl).view.bottom - ( tEHdl).view.top);
334     scBHVVal.value = ( tEHdl).dest.top - ( tEHdl).offsetV;
335 }
336
337 /* テキストエディットのレクタングル、および */
338 /* スクロールバーの位置などを更新する関数 */
339 /* sw == 0 スクロールバーが移動しなかった場合 */
340 /* != 0 スクロールバーが移動した場合 */
341 void UpdateText( int sw)
342 {
343     TMHide( tEHdl); /* テキストを不可視に */
344     SetTextRect(); /* テキストのレクタングルを計算 */
345     TMSetView( tEHdl, &viewRect); /* ビューレクタングルを設定 */
346     TMSHOW( tEHdl); /* テキストを可視に */
347
348     SetScBValue(); /* スクロールバーの値を計算 */
349     CMMinSet( scBHHdl, scBHVVal.min); /* 最小値を設定 */
350     CMMinSet( scBVHdl, scBHVVal.min);
351     CMMaxSet( scBHHdl, scBHVVal.max); /* 最大値を設定 */
352     CMMaxSet( scBVHdl, scBHVVal.max);
353     CMValueSet( scBHHdl, scBHVVal.value); /* 現在値を設定 */
354     CMValueSet( scBVHdl, scBHVVal.value);
355     if ( sw) {
356         SetScBRect(); /* スクロールバーの位置を計算 */
357         CMHide( scBHHdl); /* スクロールバーを不可視に */

```

```

358         CMHide( scBVHdl);
359         CMMove( scBHHdl, {( point_t *) ( &scBHRect)}); /* 位置を設定 */
360         CMMove( scBVHdl, {( point_t *) ( &scBVRect)});
361
362         /* サイズを設定 */
363         CMSize( scBHHdl, ( ( scBHRect.right - scBHRect.left) << 16) + 18);
364         CMSize( scBVHdl, ( (18 << 16) + ( scBVRect.bottom - scBVRect.top)));
365         CMShow( scBHHdl); /* スローハッチを可視に */
366         CMShow( scBVHdl);
367         WMDrawGBox( winPtr); /* サイズボタンを描画 */
368         SXValidScBar( winPtr); /* スローハッチをフックデモリジョンから除く */
369     }
370
371     /*
372     /* コントロールの操作に従ってテキストをスクロール */
373     /* し、スクロールバーの位置などを更新する関数 */
374     void ScrollText( point_t localPt, unsigned long shiftbit)
375     {
376         int part;
377         int destOfs[2];
378         int dir = 1;
379         control += ctrlHdl;
380
381         GMSetGraph( { graph_t } ( winPtr));
382         TMGetDestOffset( tEHdl, destOfs); /* デスティネーションタグの */
383         /* オフセットを取得 */
384         part = CMFind( localPt, winPtr, &ctrlHdl); /* 押されたコントロールを調べる */
385         if ( ( part == C_INUP) || ( part == C_INPGUP))
386             dir = -1; /* マイナス方向へのスクロール */
387         switch ( part) {
388             case C_INUP: /* アップボタン */
389             case C_INDOWN: /* ダウンボタン */
390                 if ( ( shiftbit & EHM_SHIFT) == 0) { /* シフトが押されていない */
391                     if ( ctrlHdl == scBHHdl)
392                         destOfs[ 0] += ( FONTSIZE * 8 * dir);
393                     else
394                         destOfs[ 1] += ( FONTSIZE * 2 * dir);
395                     break;
396                 } /* シフトが押されている場合次へ */
397             case C_INPGUP: /* ページアップ */
398             case C_INPGDOWN: /* ページダウン */
399                 if ( ctrlHdl == scBHHdl)
400                     destOfs[ 0] += ( { *tEHdl).view.right - ( *tEHdl).view.left) * dir;
401                 else
402                     destOfs[ 1] += ( { *tEHdl).view.right - ( *tEHdl).view.left) * dir;
403                 break;
404             case C_INTHUMB: /* サム */
405                 CMCheck( ctrlHdl, localPt, 0);
406                 if ( ctrlHdl == scBHHdl)
407                     destOfs[ 0] = CMValueGet( ctrlHdl);
408                 else
409                     destOfs[ 1] = CMValueGet( ctrlHdl);
410                 break;
411             default: /* それ以外ならば */
412                 part = 0; /* パートコードを0に */
413         }
414         if ( part != 0) {
415             /* スローハッチが操作された場合 */
416             TMUoDateExist( tEHdl, 1); /* 未アップデート部を描画して */
417             TMSetDestOffset( tEHdl, destOfs[ 0], destOfs[ 1]);
418             /* デスティネーションタグのオフセットを設定 */
419             /* !=0 ならばスクロール中 */
420             scrollContinue = part;
421         }
422     }

```

COLUMN GCC による開発

本文では XC2 による開発について述べましたが、現在 X68000 で広く使用されている C 言語処理系として、GNU C コンパイラ (GCC) についても触れておかなければならないでしょう。

GCC による SX アプリケーションの開発について述べる前に、GCC そのものについて少し解説しておきましょう。

GCC は、リチャード・ストールマン氏を中心とするフリーソフトウェアファウンデーション

ン (FSF) によって、GNU プロジェクトの一環として開発された C 言語処理系です。FSF という団体は、ソフトウェアは人類の共有の資産であることを主張し、著作権による独占に反対しています。FSF のソフトウェアは「Copyleft」という権利によって守られており、これらのソフトウェアを自由に流通させることを阻むことは固く禁止されています。原則として、欲しい人には無料で配布されなければならず、また実行コードばかりでなく、ソースを求める人にも必ず配布されなければなりません。

GNU プロジェクトというのは、OS をはじめとする環境のすべてをフリーで提供しようというもので、GCC はそのための基本的な開発言語です。もちろん、そのソースも無料で流通しているので、UNIX 系のワークステーションはもちろん、さまざまなパソコンにも、世界各国の有志の手によって移植されています*1。

X68000 版の GCC も、こうした流れの 1 つとして存在するものです。複数の方が移植を手がけられたことから、X68000 版とひと口にいても、いくつかの分派が存在するのですが、ここでは現在のところ、もっとも広く使われていると考えられる「真里子版*2」を想定して話を進めることにします。

X68000 版 GCC の特徴は、

- 1) 非常に強力な最適化が行われること
- 2) 文法は ANSI に完全対応しているが、GCC 独自の拡張も行われていること
- 3) XC (1, 2) のライブラリが使用できること*3

といった点にあります。これはすなわち、XC2 用に書いたソースがそのまま通る（ただし、XC2 の文法チェックの網の目をすり抜けたエラーは修正する必要がありますが）こと、SXLIB.L も XC2 同様に利用できること、そして、XC2 よりも効率のよいコードが得られることを意味しています。

このため、本書に掲載した C 版スケルトン、C 版のサンプルプログラムを GCC でコンパイルし、実行ファイルを作成することはもちろん、SX-WINDOW 上での実行についても XC2 と同様に行うことができます。

4 章で掲載したサンプルプログラム、CSAMPLE.C をコンパイルする場合は、次のように GCC を起動します。

```
A>gcc CSAMPLE.C SXLIB.L FLOATFNC.L
```

最後に FLOATFNC.L をリンクするよう指定していますが、これは XC2 では自動的にリンクされていたものを明示したにすぎません。FLOATFNC.L は浮動小数点演算を行うためのライブラリで、XC2 に添付されているものです。

XC2 用に記述したソースを GCC でコンパイルさせるだけでは宝の持ちぐさですから、GCC らしい、エレガントな記述で SX アプリケーションを書くことができるよう、努力してみてください。

忘れてはならないのは、適切な最適化オプションをコマンドラインで指定してこそ GCC

の最適化の威力が真に発揮される、ということです。GCCには多くの最適化オプションが用意されており、コンパイルするプログラムの性格によって、適用する最適化処理をユーザが選んで指定するようになっていきます。付属の、または別配布のドキュメントやマニュアルをよく読んで、GCCのパワーをフルに引き出してください。

真里子版GCCの最新の動向としては、SX-GCCと呼ばれる処理系ができています。グローバル変数などをワークエリアに置くなどの工夫によって、効率よくメモリが使える、SXアプリケーション開発用のコンパイラです。この原稿執筆時点（'91年11月）ではプロトタイプ版が公開されているのみですが、一日も早い完成を願いたいものです*4。

- *1:ただし、マッキントッシュに移植することは禁止されています。なにやら「Copyleft」を旨とするFSFの姿勢と相容れないものがあるのだそうで……。
- *2:真里子版GCCは、真里子氏（ハンドル名）の手によってX68000上に移植され、現在もバージョンアップが続けられています。NIFTY Serve内のSHARPフォーラム（FSHARP）で配布を受けることができます。
- *3:現在のところ、GCC用のライブラリは存在しないので、XC1、あるいはXC2のライブラリを使用する「必要がある」というのが正しい表現です。GCC自体は無料ですが、実用に供するためにはXC1、あるいはXC2を購入する必要があります。
- *4:このように、フリーソフトウェアの作者/移植者にバージョンアップを強要することはマナー違反ですから、真似をしてはいけません。

COLUMN | ライブラリのアセンブラからの使用

C言語で使用することを目的として用意されているSXLIB.Lではありますが、中にはアセンブリ言語で記述したプログラムからも利用したくなるような便利なものも含まれています。こういった便利なものはアセンブリ言語のプログラムからも利用させてもらいましょう。

例として、テキストマネージャのTMEEventWを利用することを考えてみます。

5章のリファレンスの中では、TMEEventWを呼び出す手順が、次のように示されています。

[コール]

```

pea    eventRec
pea    tEHdl
jsr    _TMEEventW
addq.l #8,sp
```

要するに、SXCALLマクロで呼び出すかわりに、jsr_TMEEventWのようにして呼び出してやればよいのです。このとき、注意が必要なのは、_TMEEventWが外部シンボルであることを宣言しておく必要があることです。ソースの先頭などで、

```
.xref _TMEventW
```

のようにして宣言しておいてください。これでアセンブルは問題なく行えるはずです。

次はリンクについて考えてみます。

このプログラムを `foo.s` とすると、アセンブルした結果、`foo.o` というオブジェクトファイルが生成されているはずです。通常、リンクは

```
A>LK SKELTON foo -Ofoo
```

といったぐあいに行いますが、ライブラリをリンクしなければならないので、次のようにリンクを行ってください。

```
A>LK SKELTON foo A:¥LIB¥SXLIB.L -Ofoo
```

外部シンボルの宣言が行われていれば、リンカが `SXLIB.L` の中から該当するルーチンを自動的にリンクしてくれるはずです。

SX コールリファレンス

SX-SYSTEM に用意されている機能を利用するには、SX コールと呼ばれる\$A 系列未定義命令を利用した手順を経て行います。この章では、SX1.10 になって追加、あるいは仕様が変更された SX コールについて、筆者が独自に解析した結果をもとにしたリファレンスを示しています。前著『SX-WINDOW プログラミング』のリファレンスとあわせてご利用ください。

SXコール・リファレンスの利用法

SX-SYSTEMの各マネージャの機能は、未定義命令\$A系列を通じてユーザーに提供されます。これら提供されるものを**SXコール**と呼ぶことにします。

SXコールの一般的な利用法は次のとおりです。

- (1)必要な数、型の引数をスタックに積む
- (2)SXコール疑似命令(\$A系列未定義命令)を実行する
- (3)スタックを補正する

(2)は、SXコール疑似命令をdc.wで置いてもかまわないのですが、SXコールであることを明示するために、マクロSXCALLを定義することにします。

SXCALL macro	num
dc.w	num
endm	

また、SXコールの名称と疑似命令コードをequで結び付けるインクルードファイルを用意し、それを利用することによって、さらに可読性が上がりますが、本書では番号で示すことにしています。

SXコールは、HumanのDOSコールなどと同様です。結果はレジスタのD0とA0に返ります。返り値を返さないSXコールでもD0とA0は破壊されるので注意してください。フラグ類も変化します。

凡例：

コール名。コール名が●になっているものは未公開コールなので、利用するのは控えたほうがよい。また、◎が記されているコールは、SX1.02からSX1.10になって仕様が変更になったものや正式に公開されたコールを表す。

コールする際にスタックに積むべき引数。そのサイズ、引数名、引数の持つ意味が書かれている。

SXコール番号

\$A092 KBFlagGet

キーボードマネージャのフラグ類を一括して返す。

コール機能解説。そのコールが持つ働きと使用上の注意が書かれている。

[引数] long kbRec キーボードレコードのアドレス
[返り値] D0.L キーボードマネージャのフラグ

bit0	Halt
bit1	ResetOn
bit2	OldOn
bit3	LedOn
bit4	ClickOn
bit5	RepeatOn
bit6	AssignOn

呼び出し後の戻り値。レジスタとその意味が書かれている。

[コール]
pea kbRec
SXCALL \$A092
addq.l #4,sp

コール例。おもに引数の順に注意してほしい。
引数のないSXコールについては載せていない。

●リファレンス利用に関する注意!

SXコールリファレンスは、筆者の独自の解析をもとに解説されています。とくに、未公開コールについては、SX-SYSTEMのバージョンアップなどによって動作しなくなる可能性があり、その利用には十分注意してください。未公開コールは、あくまで参考として掲載しています。

また、本資料の内容に関するシャープ(株)への質問、お問い合わせなどはいっさい行わないようにお願いします。

キーボードマネージャ

\$A092 KBFGet

キーボードマネージャのフラグ類を一括して返す。

[引数]

long kbRec キーボードレコードのアドレス

[返り値]

DO.L キーボードマネージャのフラグ

bit0	Halt
bit1	ResetOn
bit2	OldOn
bit3	LedOn
bit4	ClickOn
bit5	RepeatOn
bit6	AssignOn

[コール]

```
pea    kbRec
SXCALL $A092
addq.l #4,sp
```

\$A093 KBFSet

キーボードマネージャのフラグ類を一括して設定する。

[引数]

long kbRec キーボードレコードのアドレス
long flags フラグ類の状態

bit0	Halt
bit1	ResetOn
bit2	OldOn
bit3	LedOn
bit4	ClickOn
bit5	RepeatOn
bit6	AssignOn

[返り値]

DO.L 前のフラグの状態

[コール]

```
move.l #flags,-(sp)
pea    kbRec
SXCALL $A093
addq.l #8,sp
```

リソースマネージャ

\$A0ED RMResLinkGet

指定したリソースマップの次のリソースマップを得る。

[引数]

long ResMap リソースマップへのハンドル

[返り値]

DO.L 次のリソースマップへのハンドル

[コール]

```
pea    ResMap
SXCALL $A0ED
addq.l #4,sp
```

\$A0EE RMResTypeList

指定したリソースマップに登録されているタイプの数とリストを得る。リストは、タイプ名(1 ロングワード) がタイプの数だけ並んでいる構造。末尾は 0.L。リストが不要になったら廃棄する必要がある。

再配置が発生する。

[引数]

long argc タイプの数が返るバッファ(1 ロングワード)のアドレス
long argv タイプのリストへのハンドルが返る バッファ(1 ロング

		ワード)アドレス
long	ResMap	リソースマップのハンドル
[返回值]		
DO.L	=0	正常終了
	≠0	エラー
[コール]		
	pea	ResMap
	pea	argv
	pea	argc
	SXCALL	\$AOEF
	lea	12(sp),sp

\$AOEF RMResIDList

指定したリソースマップに登録されているタイプのIDの数とリストを得る。リストはID (1ワード) がIDの数だけ並んでいる構造。末尾はO.W。リストが不要になったら廃棄する必要がある。

再配置が発生する。

[引数]		
long	argc	IDの数が返るバッファ(1ロングワード)のアドレス
long	argv	IDのリストが返るバッファ(IDの数×1ワード)のアドレス
long	ResMap	リソースマップへのハンドル
long	Type	タイプ
[返回值]		
DO.L	=0	正常終了
	≠0	エラー
[コール]		
	move.l	# Type,-(sp)
	pea	ResMap
	pea	argv
	pea	argc
	SXCALL	\$AOEF
	lea	16(sp),sp

ウィンドウマネージャ

\$A1FF WMSelect2

winPtrで指定したウィンドウを、サブウィンドウを消去せずにアクティブにする。

再配置が発生する。

[引数]		
long	winPtr	ウィンドウレコードのアドレス
[返回值]		
DO.L	リザルトコード	
[コール]		
	pea	winPtr
	SXCALL	\$A1FF
	addq.l	# 4,sp

\$A22C WMOptionGet

カレントウィンドウのウィンドウオプション(wOption)を返す。

[引数]		
	なし	

[返回值]		
DO.L	ウィンドウオプション(下位ワードのみ意味を持つ)	

\$A22D WMOptionSet

カレントウィンドウのウィンドウオプション(wOption)を設定する。

[引数]		
word	wOpt	ウィンドウオプション
[返回值]		
なし		
[コール]		
	move.w	# wOpt,-(sp)
	SXCALL	\$A22D
	addq.l	# 2,sp

コントロールマネージャ

\$A2A0 CMOptionGet

ctrlHdlで指定したコントロールの、コントロールオプション (cOption) を返す。

[引数]

long ctrlHdl コントロールレコードへのハンドル

[返り値]

DO.L コントロールオプション(下位ワードの意味を持つ)

[コール]

```
pea      ctrlHdl
SXCALL   $A2A0
addq.l    # 4,sp
```

\$A2A1 CMOptionSet

ctrlHdlで指定したコントロールの、コントロールオプション (cOption) を設定する。

[引数]

long ctrlHdl コントロールレコードへのハンドル
word cOpt コントロールオプション

[返り値]

なし

[コール]

```
move.w    # cOpt,-(sp)
pea      ctrlHdl
SXCALL   $A2A1
addq.l    # 6,sp
```

\$A2A2 CMUserGet

ctrlHdlで指定したコントロールの、ユーザ用のワーク (cUser) を返す。

[引数]

long ctrlHdl コントロールレコードへのハンドル

[返り値]

DO.L ユーザ用のワークの値

[コール]

```
pea      ctrlHdl
SXCALL   $A2A2
addq.l    # 4,sp
```

\$A2A3 CMUserSet

ctrlHdlで指定したコントロールの、ユーザ用のワーク (cUser) を設定する。

[引数]

long ctrlHdl コントロールレコードへのハンドル
long cUser ユーザ用のワークの値

[返り値]

なし

[コール]

```
move.l    # cUser,-(sp)
pea      ctrlHdl
SXCALL   $A2A3
addq.l    # 8,sp
```

\$A2A4 CMProcGet

ctrlHdlで指定したコントロールの、ドラッグ時の手続きのアドレス (cProc) を返す。

[引数]

long ctrlHdl コントロールレコードへのハンドル

[返り値]

DO.L ドラッグ時の手続きのアドレス

[コール]

```
pea      ctrlHdl
SXCALL   $A2A4
addq.l    # 4,sp
```

\$A2A5 CMProcSet

ctrlHdlで指定したコントロールの、ドラッグ時の手続きのアドレス (cProc) を設定する。

[引数]

long ctrlHdl コントロールレコードへのハンドル
long cProc ドラッグ時の手続きのアドレス

[返り値]

なし

[コール]

```

    pea    cProc
    pea    ctrlHdl
    SXCALL $A2A5
    addq.l #8,sp

```

\$A2A6 CMDefDataSet

ctrlHdlで指定したコントロールの、定義関数のデータ (cDefData) を返す。

[引数]

```

    long    ctrlHdl    コントロールレコードへのハ
                        ンドル

```

[返り値]

```

    DO.L    定義関数のデータ

```

[コール]

```

    pea    ctrlHdl
    SXCALL $A2A6

```

```

    addq.l #4,sp

```

\$A2A7 CMDefDataSet

ctrlHdlで指定したコントロールの、定義関数のデータ (cDefData) を設定する。

[引数]

```

    long    ctrlHdl    コントロールレコードへのハ
                        ンドル
    long    cDefData    定義関数のデータ

```

[返り値]

なし

[コール]

```

    move.l #cDefData,-(sp)
    pea    ctrlHdl
    SXCALL $A2A7
    addq.l #8,sp

```

メニューマネージャ

\$A269 MNConvert

strZPtrで指定した文字列によって、メニューレコードを作成する。menuHdlが0の場合、メニューマネージャがヒープ上に作成する。

メニュー定義文字列は、基本的にメニューアイテムを1つずつカンマで区切ったもので、特殊文字を利用することによって、ショートカットやチェックマークの指定を行うことができる。

特殊文字	内 容
^	ショートカット文字の指定。次の1文字がショートカット文字となる
-	この文字で始まるアイテムはインアクティブとなる
!	チェックマークをつける

再配置が発生する。

[引数]

```

    long    menuHdl    メニューレコードへのハンドル
    long    strZPtr     メニュー定義文字列(ASCIIIZ)
                        へのポインタ
    word    Id          メニュー定義関数の ID

```

[返り値]

```

    DO.L    リザルトコード
    AO.L    メニューレコードへのハンドル

```

[コール]

```

    move.w #Id,-(sp)
    pea    strZPtr
    pea    menuHdl
    SXCALL $A269
    lea    10(sp),sp

```


サブウィンドウマネージャ

\$A227 WSOpen

新しいサブウィンドウを開く。sWinPtr が 0 の場合、サブウィンドウマネージャがヒープ上に作成する。再配置が発生する。

[引数]

long	sWinPtr	サブウィンドウレコードのアドレス
long	rgnHdl	アウトサイドリージョンとなるリージョンへのハンドル
long	prio	プライオリティ値

[返り値]

DO.L	リザルトコード
AO.L	サブウィンドウレコードのアドレス

[コール]

```

move.l    #prio, -(sp)
pea       rgnHdl
pea       sWinPtr
SXCALL    $A227
lea       12(sp), sp

```

\$A228 WSClose

sWinPtr で指定したサブウィンドウを閉じ、サブウィンドウリストから削除する。サブウィンドウレコードをヒープ上以外に作成していた場合に使用する。

再配置が発生する。

[引数]

long	sWinPtr	サブウィンドウレコードのアドレス
------	---------	------------------

[返り値]

DO.L	リザルトコード
------	---------

[コール]

```

pea       sWinPtr
SXCALL    $A228
addq.l    #4, sp

```

\$A229 WSDispose

sWinPtr で指定したサブウィンドウを閉じ、サブウィンドウリストから削除した後、サブウィンドウレ

コードとして確保されていたブロックを廃棄する。サブウィンドウレコードをヒープ上に作成していた場合に使用する。

再配置が発生する。

[引数]

long	sWinPtr	サブウィンドウレコードのアドレス
------	---------	------------------

[返り値]

DO.L	リザルトコード
------	---------

[コール]

```

pea       sWinPtr
SXCALL    $A229
addq.l    #4, sp

```

\$A22A WSEnlist

sWinPtr で指定したサブウィンドウをサブウィンドウリストに加える。

[引数]

long	sWinPtr	サブウィンドウレコードのアドレス
------	---------	------------------

[返り値]

DO.L	リザルトコード
------	---------

[コール]

```

pea       sWinPtr
SXCALL    $A22A
addq.l    #4, sp

```

\$A22B WSDelist

sWinPtr で指定したサブウィンドウをサブウィンドウリストから削除する。

[引数]

long	sWinPtr	サブウィンドウレコードのアドレス
------	---------	------------------

[返り値]

DO.L	リザルトコード
------	---------

[コール]

```

pea       sWinPtr
SXCALL    $A22B
addq.l    #4, sp

```

プリントマネージャ

\$A4E0 PMinIt

プリントマネージャを初期化する。

メモリマネージャ、リソースマネージャ、イベントマネージャ、メニューマネージャ、グラフィックマネージャ、ウィンドウマネージャが初期化され、リソースファイル SYSTEM.LB がオープンされている必要がある。

[引数]

なし

[返回值]

DO.L リザルトコード

\$A4E1 PMTini

プリントマネージャの終了処理を行う。

再配置が発生する。

[引数]

なし

[返回值]

DO.L リザルトコード

\$A4E2 PMOpen

drvIDで指定したIDのプリンタドライバをリソースPTRDからメモリ上に読み込みロックする。drvIDとして-1を指定した場合、SRAMに記録されているデフォルトのプリンタドライバが使用される。

再配置が発生する。

[引数]

word drvID ドライバのID

[返回值]

DO.L リザルトコード

= -2 すでにドライバがオープンされている

[コール]

move.w #drvID, -(sp)

SXCALL \$A4E2

addq.l #2, sp

\$A4E3 PMClose

プリンタドライバの終了処理を行い、ドライバが使用していたブロックを廃棄する。

再配置が発生する。

[引数]

なし

[返回值]

DO.L リザルトコード

\$A4E4 PMSetDefault

prRecHdlで指定した印刷環境レコードにデフォルトの値(リソースPrEVのIDOに記録されている、あるいはドライバ自体が保持している)をセットする。

再配置が発生する。

[引数]

long prRecHdl 印刷環境レコードへのハンドル

[返回值]

DO.L リザルトコード

AO.L 印刷環境レコードへのハンドル

[コール]

pea prRecHdl

SXCALL \$A4E4

addq.l #4, sp

\$A4E5 PMValidate

prRecHdlで指定した印刷環境レコードの内容が正しいかどうかをチェックし、調整する。

[引数]

long prRecHdl 印刷環境レコードへのハンドル

[返回值]

DO.L = 0 調整せず、レコードの内容に変化はない
= 1 調整を行った
= -1 エラー

[コール]

pea prRecHdl

SXCALL \$A4E5

addq.l #4, sp

\$A4E6 PMImageDialog

ページ印刷用の印刷環境設定ダイアログをオープンし、ユーザの操作を受けつけた後、クローズする。その結果をもとに prRecHdl で指定した印刷環境レコードの内容を設定する。

再配置が発生する。

[引数]

long prRecHdl 印刷環境レコードへのハンドル

[返り値]

DO.L =0 レコードの内容に変化はない
 =1 設定を行った
 =-1 エラー

[コール]

```
pea prRecHdl
SXCALL $A4E6
addq.l #4,sp
```

\$A4E7 PMStrDialog

コード印刷用の印刷環境設定ダイアログをオープンし、ユーザの操作を受けつけた後、クローズする。その結果をもとに prRecHdl で指定した印刷環境レコードの内容を設定する。

再配置が発生する。

SX1.10 標準添付のプリンタドライバではサポートされていない。

[引数]

long prRecHdl 印刷環境レコードへのハンドル

[返り値]

DO.L =0 レコードの内容に変化はない
 =1 設定を行った
 =-1 エラー

[コール]

```
pea prRecHdl
SXCALL $A4E7
addq.l #4,sp
```

\$A4E9 PMEnvCopy

srcHdl で指定した印刷環境レコードの内容を、dstHdl で指定したレコードにコピーする。その際、値のチェックと調整が行われる。

[引数]

long srcHdl コピー元の印刷環境レコードへのハンドル
 long dstHdl コピー先の印刷環境レコードへのハンドル

[返り値]

DO.L =0 調整を行わなかった
 =1 調整を行った
 =-1 エラー

[コール]

```
pea dstHdl
pea srcHdl
SXCALL $A4E9
addq.l #8,sp
```

\$A4EA PMJobCopy

srcHdl で指定した印刷環境レコードの実行部分のデータ dstHdl で指定したレコードにコピーする。その際、値のチェックと調整が行われる。

実行部分とは、具体的には印刷開始ページ (prFstPage)、印刷終了ページ (prLstPage)、1 ページあたりの印刷枚数 (prDupPage)、印刷モード (prMode)、印刷モードのマスク (prMask)、そして、システム予約 (prJobRsv) を意味する。

[引数]

long srcHdl コピー元の印刷環境レコードへのハンドル
 long dstHdl コピー先の印刷環境レコードへのハンドル

[返り値]

DO.L =0 調整を行わなかった
 =1 調整を行った
 =-1 エラー

[コール]

```
pea dstHdl
pea srcHdl
SXCALL $A4EA
addq.l #8,sp
```

\$A4EB PMOpenImage

ページ印刷用のグラフポートを作成し、ページ印刷を開始する。

再配置が発生する。

[引数]

long prRecHdl 印刷環境レコードへのハンドル

[返り値]

DO.L リザルトコード
AO.L グラフポートのアドレス

[コール]

```
pea      prRecHdl
SXCALL   $A4EB
addq.l   #4,sp
```

\$A4EC PMRecordPage

ページ印刷のスキプトの記録を開始する。
rectPtr で指定した範囲が、後の印刷時に印刷される。

再配置が発生する。

[引数]

long rectPtr 印刷範囲を意味するレクタングルレコードのアドレス

[返り値]

DO.L リザルトコード

[コール]

```
pea      rectPtr
SXCALL   $A4EC
addq.l   #4,sp
```

\$A4ED PMPrintPage

ページ印刷用スキプトの記録を終了し、実際の印刷を開始する。

再配置が発生する。

[引数]

long param かならず 0 を指定する

[返り値]

DO.L リザルトコード

[コール]

```
move.l    #param, -(sp)
SXCALL    $A4ED
addq.l    #4,sp
```

\$A4EE PMCancelPage

ページ印刷用スキプトの記録を中止する。印刷は行われない。

再配置が発生する。

[引数]

なし

[返り値]

DO.L リザルトコード

\$A4EF PMAction

印刷処理を行う。ctrl で指定した動作を行い、その結果を返す。

再配置が発生する。

[引数]

word ctrl 動作の指定

0(PC_STAT)	印刷を続行する
1(PC_END)	印刷を終了する
2(PC_STOP)	印刷を中断する
3(PC_CONT)	印刷を再開する

[返り値]

DO.L 実行結果

0(P_FINISH)	印刷が終了した
1(P_WORKING)	印刷中
2(P_RESTING)	印刷を中断した
3(P_TIMEOUT)	タイムアウト発生
-1(P_ERROR)	エラー発生

[コール]

```
move.w    #ctrl, -(sp)
SXCALL    $A4EF
addq.l    #2,sp
```

\$A4F0 PMCloselImage

ページ印刷を終了し、グラフポートなどを廃棄する。
再配置が発生する。

[引数]

なし

[返り値]

DO.L リザルトコード

\$A4F1 PMDrawString

strHdl, length で指定した文字列のコード印刷を開始する。

再配置が発生する。

[引数]

long prRecHdl 印刷環境レコードへのハンドル
long strHdl 文字列へのハンドル

long length 文字列のバイト数
 long strOpt 印刷オプション
 =0 印刷終了時に改ページ
 コードを出力する
 =1 印刷終了時に改ページ
 コードを出力しない

[返り値]

DO.L リザルトコード

[コール]

```
move.l    # strOpt, -(sp)
move.l    # length, -(sp)
pea       strHdl
pea       prRecHdl
SXCALL    $A4F1
lea       16(sp), sp
```

\$A4F2 PMVer

プリントマネージャのバージョンを返す。

[引数]

なし

[返り値]

DO.L バージョン番号(バージョン1.00で\$0100)

\$A4F3 PMDrvVer

現在オープンされているプリンタドライバのバージョンを返す。

[引数]

なし

[返り値]

DO.L バージョン番号(バージョン1.00で\$0100)
 =-1 プリンタドライバがオープン
 されていない

\$A4F4 PMDrvCtrl

プリンタドライバを直接制御する。プリンタドライバに与えるコマンド、パラメータについては本文参照。

[引数]

long cmd ドライバに与えるコマンド
 long param1 パラメータ 1
 long param2 パラメータ 2
 long param3 パラメータ 3

[返り値]

DO.L リザルトコード

[コール]

```
move.l    # param3, -(sp)
move.l    # param2, -(sp)
move.l    # param1, -(sp)
move.l    # cmd, -(sp)
SXCALL    $A4F4
lea       16(sp), sp
```

\$A4F5 PMDrvID

現在オープンされているプリンタドライバの ID を返す。

[引数]

なし

[返り値]

DO.L プリンタドライバの ID(下位ワードのみ意味を持つ)
 =-1 プリンタドライバがオープン
 されていない

\$A4F6 PMDrvHdl

現在オープン中のプリンタドライバが収められているブロックへのハンドルを返す。

[引数]

なし

[返り値]

DO.L リザルトコード
 =-1 プリンタドライバがオープン
 されていない
 AO.L プリンタドライバへのハンドル

\$A4F7 PMMaxRect

pKindで指定した用紙の印刷可能な最大範囲を、rectPtrで指定したレクタングルレコードに返す。

[引数]

long prRecHdl 印刷環境レコードへのハンドル
 word pKind 用紙の種類
 long rectPtr 結果が返るレクタングルレコードのアドレス

[返り値]

DO.L リザルトコード
 AO.L 印刷環境レコードへのハンドル

[コール]

```
pea       rectPtr
```

```

move.w    # pKind, -(sp)
pea       prRecHdl
SXCALL    $A4F7
lea       10(sp), sp

```

\$A4F8 PMSaveEnv

prRecHdlで指定した印刷環境レコードの内容をデフォルトの値として、リソース PrEV の IDO に記録する。

再配置が発生する。

[引数]

long prRecHdl 印刷環境レコードへのハンドル

[返り値]

DO.L リザルトコード

[コール]

```

pea       prRecHdl
SXCALL    $A4F8
addq.l    # 4, sp

```

\$A4F9 PMReady

プリンタの状態を調べ、結果を返す。

[引数]

なし

[返り値]

DO.L プリンタの状態

0(PS_BUSY)	印刷不可
1(PS_READY)	印刷可

\$A4FA PMProcPrint

procPtrで指定したユーザープロセスを登録し、プロセス印刷を開始する。

[引数]

long prRecHdl 印刷環境レコードへのハンドル
long procPtr ユーザープロセスのアドレス

[返り値]

DO.L リザルトコード

[コール]

```

pea       procPtr
pea       prRecHdl
SXCALL    $A4FA
addq.l    # 8, sp

```

\$A4FB PMDrvInfo

drvIDで指定したプリンタドライバに関する情報を、resultPtrで指定したバッファに返す。drvIDとして-1を指定すると、デフォルトのプリンタドライバの情報が返る。

情報の形式は以下のとおり。

+\$00.w	プリンタドライバの ID
+\$02.w	プリンタドライバのバージョン
+\$04	プリンタ名 (ASCIIZ)

[引数]

word drvID プリンタドライバの ID
long resultPtr 結果が返るバッファのアドレス

[返り値]

DO.L リザルトコード

AO.L resultPtr

[コール]

```

pea       resultPtr
move.w    # drvID, -(sp)
SXCALL    $A4FB
addq.l    # 6, sp

```

テキストマネージャ

\$A317 ◎TMKey.

tEHdlで指定されたテキストエディットレコードの編集テキストに、keyCharで指定されたキャラクタを入力して再表示する。あらかじめグラフポートをセットしておく必要がある。

再配置が発生する。

[引数]

long tEHdl テキストエディットレコードへのハンドル
word keyChar 入力キャラクタ

[返り値]

DO.L リザルトコード
=0 編集しなかった
=1 編集した

[コール]

```
move.w    #keyChar,-(sp)
pea       tEHdl
SXCALL    $A317
addq.l    #6,sp
```

\$A318 ◎TMStr

tEHdlで指定されたテキストエディットレコードの編集テキストのセレクト領域を、textPtrで指定した文字列と置き換える。CR (\$0D) 以外の制御コードを含めることはできない。再表示は行わない。

再配置が発生する。

[引数]

long tEHdl テキストエディットレコードへのハンドル
long textPtr テキストへのポインタ
long length テキストのバイト数

[返り値]

DO.L リザルトコード
=0 編集しなかった
=1 編集した

[コール]

```
move.l    #length,-(sp)
pea       textPtr
pea       tEHdl
```

```
SXCALL    $A318
lea       12(sp),sp
```

\$A319 TMCalText

tEHdlで指定したテキストエディットの段落情報を計算/設定する。カーソルの位置などは変更されない。通常は\$A464 TMSetSelCalを利用する。再表示は行われない。

再配置が発生する。

[引数]

long tEHdl テキストエディットレコードへのハンドル

[返り値]

DO.L リザルトコード

[コール]

```
pea       tEHdl
SXCALL    $A319
addq.l    #4,sp
```

\$A31C ◎TMEvent

tEHdlで指定されたテキストエディットレコードについて、eventRecで指定されたイベントレコードの内容に対応する処理を行う。あらかじめグラフポートをセットしておく必要がある。

再配置が発生する。

対応するのは、以下の4つのイベント。これらの処理の後、再表示が行われる。

- ・スルイベント
キャレットの点滅を行う。
- ・マウスレフトダウンイベント
セレクト領域の変更を行う。
- ・マウスライトダウンイベント
ポップアップメニューを表示して、テキストマネージャスクラップとの間でカット&ペーストを行う。
- ・キーダウンイベント
キャラクタの入力を行う（全角にも対応）。カーソルキー、BS、DEL、ROLL UP/DOWN キーに対応する。

[引数]

long tEHdl テキストエディットレコード

		へのハンドル
long	eventRec	イベントレコードのアドレス
[返り値]		
DO.L	リザルトコード	
	= 0	編集しなかった
	= 1	編集した

```
[コール]
    pea    eventRec
    pea    tEHdl
    SXCALL $A31C
    addq.l  #8,sp
```

\$A320 ©TMCut

↑EHD1で指定されたテキストエディットレコードの編集テキストのセレクト領域をカットし、テキストマネージャスクラップに移し、再表示する。あらかじめグラフポートをセットしておく必要がある。

再配置が発生する。

[引数]

long	tEHdl	テキストエディットレコード へのハンドル
------	-------	-------------------------

[返り値]

DO.L	リザルトコード
=0	編集しなかった
=1	編集した

```

[コール]
        pea        tEHdl
        SXCALL     $A320
        addq.l     #4,sp

```

\$A322 ©TMPaste

tEHdlで指定されたテキストエディットレコードの編集テキストのセレクト領域を、テキストマネージャスクラップの内容と置き換え、再表示する。あらかじめグラフポートをセットしておく必要がある。テキストマネージャスクラップの内容は変化しない。

再配置が発生する。

[引数]		
long	tEHdl	テキストエディットレコード へのハンドル

[返り値]

DO.L	リザルトコード
=0	編集しなかった
=1	編集した

[コール]

```

pea      tEHdl
SXCALL   $A322
addq.l   # 4,sp

```

\$A323 ©TMDDelete

TEHD1で指定されたテキストエディットレコードの編集テキストのセレクト領域をカットし、再表示する。テキストマネージャスクラップの内容は変化しない。あらかじめグラフポートをセットしておく必要がある。

再配置が発生する。

[引数]

long	tEHdl	テキストエディットレコード へのハンドル
------	-------	-------------------------

[返り値]

DO.L リザルトコード
 =0 編集しなかった
 =1 編集した

[コール]

```

pea      tEHdl
SXCALL   $A324
addq.l   # 4.sp

```

\$A324 TMinser

tEHdlで指定したテキストエディットのセレクト領域と strPtr で指定した文字列を置き換えて再表示する。

再配置が発生する。

[引数]

long	tEHdl	テキストエディットレコード へのハンドル
------	-------	-------------------------

long	strPtr	文字列のアドレス
long	length	文字列のバイト数

[返り値]

D0.L リザルトコード
 =0 編集しなかった
 =1 編集した

[コール]

```
move.l    #length,-(sp)
pea       strPtr
pea       tEHdl
SXCALL    $A324
lea       l2(sp),sp
```

\$A32C TMCacheON

tEHd1で指定したテキストエディットについて、sizeで指定したサイズのキャッシュを用意し、ONにする。

再配置が発生する。

[引数]

long tEHd1 テキストエディットレコードへのハンドル

long size キャッシュのバイト数

[返り値]

DO.L リザルトコード

[コール]

move.l #size,-(sp)

pea tEHd1

SXCALL \$A32C

addq.l #8,sp

\$A32D TMCacheOFF

tEHd1で指定したテキストエディットレコードのキャッシュを廃棄し、OFFにする。

再配置が発生する。

[引数]

long tEHd1 テキストエディットレコードへのハンドル

[返り値]

DO.L リザルトコード

[コール]

pea tEHd1

SXCALL \$A32D

addq.l #4,sp

\$A32E TMCacheFlush

tEHd1で指定したテキストエディットレコードのキャッシュをフラッシュする。

再配置が発生する。

[引数]

long tEHd1 テキストエディットレコードへのハンドル

[返り値]

DO.L リザルトコード

[コール]

pea tEHd1

SXCALL \$A32E

addq.l #4,sp

\$A32F TMShow

tEHd1で指定したテキストエディットのドローレベルを+1する。再描画は行わない。

[引数]

long tEHd1 テキストエディットレコードへのハンドル

[返り値]

DO.L ドローレベルを+1した結果の値

[コール]

pea tEHd1

SXCALL \$A32F

addq.l #4,sp

\$A330 TMHide

tEHd1で指定したテキストエディットのドローレベルを-1する。再描画は行わない。

[引数]

long tEHd1 テキストエディットレコードへのハンドル

[返り値]

DO.L ドローレベルを-1した結果の値

[コール]

pea tEHd1

SXCALL \$A330

addq.l #4,sp

\$A331 TMSelShow

tEHd1で指定したテキストエディットのハイライト表示レベルを+1する。再描画は行わない。

[引数]

long tEHd1 テキストエディットレコードへのハンドル

[返り値]

DO.L ハイライト表示レベルを+1した結果の値

[コール]

pea tEHd1

SXCALL \$A331

addq.l #4,sp

\$A332 TMSelHide

tEHdlで指定したテキストエディットのハイライト表示レベルを-1する。再描画は行わない。

[引数]

long tEHdl テキストエディットレコードへのハンドル

[返り値]

DO.L ハイライト表示レベルを-1した結果の値

[コール]

```

pea    tEHdl
SXCALL $A332
addq.l #4,sp

```

\$A333 TMSearchStrF

tEHdlで指定したテキストエディットの offset1, offset2で示される範囲で, strPtrで指定した文字列を前方検索する。大文字小文字等は区別される。

procPtrで指定するフィルタプロセスは, 検索処理中適当な間隔で呼び出される。引数として, AOに tEHdlが渡される。返り値として DOに 0以外を返すと, 検索は中断され, TMSerachStrFの返り値として, フィルタプロセスの返り値がアプリケーションに返される。

再配置が発生する。

[引数]

long tEHdl テキストエディットレコードへのハンドル

long strPtr 検索文字列のアドレス

long length 検索文字列のバイト数

long offset1 テキストの検索開始位置(オフセット)

long offset2 テキストの検索終了位置(オフセット)

long procPtr フィルタプロセスのアドレス (0で指定しない)

[返り値]

DO.L 発見した文字列の位置(オフセット)

=-1 発見できなかった

AO.L 発見した文字列のバイト数

[コール]

```

pea    procPtr
move.l #offset2,-(sp)
move.l #offset1,-(sp)

```

```

move.l #length,-(sp)
pea    strPtr
pea    tEHdl
SXCALL $A333
lea    24(sp),sp

```

\$A334 TMSearchStrB

tEHdlで指定したテキストエディットの offset1, offset2で示される範囲で, strPtrで指定した文字列を後方検索する。大文字小文字等は区別される。

procPtrで指定するフィルタプロセスは, 検索処理中適当な間隔で呼び出される。引数として, AOに tEHdlが渡される。返り値として DOに 0以外を返すと, 検索は中断され, TMSerachStrBの返り値としてフィルタプロセスの返り値がアプリケーションに返される。

再配置が発生する。

[引数]

long tEHdl テキストエディットレコードへのハンドル

long strPtr 検索文字列のアドレス

long length 検索文字列のバイト数

long offset1 テキストの検索開始位置(オフセット)

long offset2 テキストの検索終了位置(オフセット)

long procPtr フィルタプロセスのアドレス (0で指定しない)

[返り値]

DO.L 発見した文字列の位置(オフセット)

=-1 発見できなかった

AO.L 発見した文字列のバイト数

[コール]

```

pea    procPtr
move.l #offset2,-(sp)
move.l #offset1,-(sp)
move.l #length,-(sp)
pea    strPtr
pea    tEHdl
SXCALL $A334
lea    24(sp),sp

```

\$A335 TMTextInWidth2

tEHdlで指定したテキストエディットレコードの環境下で, StartPointで指定した位置から strPtr,

offsetで指定される文字列を描画する場合、Widthで指定したドット数の幅に収まる文字数を計算し、結果を返す。

\$A197 GMStrLengthとはコントロールコードの処理が異なる。\$OA\$OD (改行コード) があった場合、そこまでで文字数の計算を終了し、\$O9 (TAB コード) があった場合、TAB サイズ (teTabSize) にしたがってタブの処理を行う。また、ほかのコントロールコードの場合、編集モード (teDrawMode) にしたがって計算を行う。

再配置が発生する。

[引数]

long	tEHdl	テキストエディットレコードへのハンドル
long	strPtr	文字列のアドレス
long	offset	文字列のオフセット
word	Width	文字列を収める幅(ドット数)
word	StartPoint	文字列のローカル座標-ディスティネーションレクタングルの水平方向オフセット

[返り値]

DO.L	収まる文字列のバイト数/リザルトコード
AO.L	=0 改行コード以外の理由で終了した
	≠0 改行コードにより終了した

[コール]

```

move.w    #StartPoint,-(sp)
move.w    #Width,-(sp)
move.l    #offset,-(sp)
pea       strPtr
pea       tEHdl
SXCALL    $A335
lea       16(sp),sp

```

\$A336 TMTextWidth2

tEHdlで指定したテキストエディットレコードの環境下で、StartPointで指定した位置から strPtr, offset, lengthで指定される文字列を描画する場合、それが占める幅(ドット数)を計算し、結果を返す。

\$A196 GMStrWidthとはコントロールコードの処理が異なる。\$OA\$OD (改行コード) があった場合、そこまででドット数の計算を終了し、\$O9 (TAB コード) があった場合、TAB サイズ (teTabSize) にしたがってタブの処理を行う。また、ほかのコントロールコードの場合、編集モード (teDrawMode) にし

たがって計算を行う。

再配置が発生する。

[引数]

long	tEHdl	テキストエディットレコードへのハンドル
long	strPtr	文字列のアドレス
long	offset	文字列のオフセット
word	length	文字列のバイト数
word	StartPoint	文字列のローカル座標-ディスティネーションレクタングルの水平方向オフセット

[返り値]

DO.L	文字列の占める幅(ドット数)/リザルトコード
------	------------------------

[コール]

```

move.w    #StartPoint,-(sp)
move.w    #length,-(sp)
move.l    #offset,-(sp)
pea       strPtr
pea       tEHdl
SXCALL    $A336
lea       16(sp),sp

```

\$A337 TMDrawText2

tEHdlで指定したテキストエディットレコードの環境下で、StartPointで指定した位置から strPtr, offset, lengthで指定される文字列を描画する。あらかじめビューレクタングルでクリップ領域を設定しておく必要がある。

\$A196 GMStrWidthとはコントロールコードの処理が異なる。\$OA\$OD (改行コード) があった場合、そこまでで描画を終了し、\$O9 (TAB コード) があった場合、TAB サイズ (teTabSize) にしたがってタブの処理を行う。また、ほかのコントロールコードの場合、編集モード (teDrawMode) にしたがって描画する。

再配置が発生する。

[引数]

long	tEHdl	テキストエディットレコードへのハンドル
long	strPtr	文字列のアドレス
long	offset	文字列のオフセット
word	length	文字列のバイト数
word	StartPoint	文字列のローカル座標-ディ

スティネーションレクタング
ルの水平方向オフセット
word TabMode =0 タブはベン位置を移動
するだけ
≠0 タブはバックグラウンド
カラーによる塗り潰し

[返り値]

DO.L リザルトコード

[コール]

```
move.w    # TabMode,-(sp)
move.w    # StartPoint,-(sp)
move.w    # length,-(sp)
move.l    # offset,-(sp)
pea       strPtr
pea       tEHdl
SXCALL    $A337
lea       18(sp),sp
```

\$A338 TMUpdate2

hisRecPtrで指定した編集履歴レコードにした
がって、tEHdlで指定したテキストエディットの描
画を行う。

再配置が発生する。

[引数]

long tEHdl テキストエディットレコード
へのハンドル

long hisRecPtr 編集履歴レコードのアドレス

[返り値]

DO.L リザルトコード

[コール]

```
pea       hisRecPtr
pea       tEHdl
SXCALL    $A338
addq.l    #8,sp
```

\$A339 TMUpdate3

tEHdlで指定した、テキストエディットの
updtRectPtrで指定したレクタングルで示される範
囲に表示される部分を再表示する。\$A313
TMUpdateとの違いは、バックグラウンドカラーに
よる塗り潰しを行わない点。

再配置が発生する。

[引数]

long tEHdl テキストエディットレコード

へのハンドル
long updtRectPtr レクタングルレコードへのポ
インタ(ローカル座標)

[返り値]

DO.L リザルトコード

[コール]

```
pea       updtRectPtr
pea       tEHdl
SXCALL    $A339
addq.l    #8,sp
```

\$A33A TMCaCOLine

tEHdlで指定したテキストエディットの、offset
で指定した段落位置 (段落の通し番号) の段落情報を
columnPtrからのバッファに作成する。

再配置が発生する。

[引数]

long tEHdl テキストエディットレコード
へのハンドル

long columnPtr 段落情報を作成するバッファ
(\$18 バイト)のアドレス

long offset 段落位置

[返り値]

DO.L リザルトコード

[コール]

```
move.l    # offset,-(sp)
pea       columnPtr
pea       tEHdl
SXCALL    $A33A
lea       12(sp),sp
```

\$A33C TMCaLine

tEHdlで指定したテキストエディットの、offset
で指定した行位置 (mode=1 の場合)、あるいはバ
イト位置 (mode=0 の場合) の段落情報を、columnPtr
からのバッファに作成する。

再配置が発生する。

[引数]

long tEHdl テキストエディットレコード
へのハンドル

long columnPtr 段落情報を作成するバッファ
(\$18 バイト)のアドレス

long offset 行位置/バイト位置

word mode =0 offsetはバイト位置

=1 offset は行位置

[返り値]

DO.L この段落のバイト位置/リザルトコード

[コール]

```
move.w    # mode,-(sp)
move.l    # offset,-(sp)
pea       columnPtr
pea       tEHdl
SXCALL    $A33C
lea       14(sp),sp
```

\$A33D TMLeftSel

tEHdlで指定したテキストエディットのセレクト領域の直前 (1 つ左) の位置を返す。

[引数]

long tEHdl テキストエディットレコードへのハンドル

[返り値]

DO.L オフセット
=-1 セレクト領域がテキストの先頭

[コール]

```
pea       tEHdl
SXCALL    $A33D
addq.l    # 4,sp
```

\$A33E TMRightSel

tEHdlで指定したテキストエディットのセレクト領域の直後 (1 つ右) の位置を返す。

[引数]

long tEHdl テキストエディットレコードへのハンドル

[返り値]

DO.L オフセット
=-1 セレクト領域がテキストの最後

[コール]

```
pea       tEHdl
SXCALL    $A33E
addq.l    # 4,sp
```

\$A33F TMPointSel

tEHdlで指定されたテキストエディットのセレクト位置を, xpoint, ypointで指定した座標によって変更し, 再表示する。セレクト領域変更後にカーソル

位置の再計算は行¹うが, スクロールは行²わない。
再配置³が発生する。

[引数]

long tEHdl テキストエディットレコードへのハンドル
long xpoint 水平座標(ローカル座標)
long ypoint 垂直座標(ローカル座標)
word extend =0 新規セレクト領域
=1 前回のセレクト領域を変更
=2 セレクト領域開始位置の変更
=3 セレクト領域終了位置の変更

[返り値]

DO.L リザルトコード

[コール]

```
move.w    # extend,-(sp)
move.l    # ypoint,-(sp)
move.l    # xpoint,-(sp)
pea       tEHdl
SXCALL    $A33F
lea       14(sp),sp
```

\$A340 TMOffsetSel

tEHdlで指定されたテキストエディットのセレクト位置を, offsetで指定したバイト位置によって変更し, 再表示する。セレクト領域変更後にカーソル位置の再計算は行¹うが, スクロールは行²わない。
再配置³が発生する。

[引数]

long tEHdl テキストエディットレコードへのハンドル
long offset バイト位置
word extend =0 新規セレクト領域
=1 前回のセレクト領域を変更
=2 セレクト領域開始位置の変更
=3 セレクト領域終了位置の変更

[返り値]

DO.L リザルトコード

[コール]

```

move.w    # extend,-(sp)
move.l    # offset,-(sp)
pea       tEHdl
SXCALL    $A340
lea       10(sp),sp

```

\$A341 TmPointToLine

tEHdlで指定したテキストエディットの、xpoint, ypointで指定した座標の段落情報を、columnPtrからのバッファに作成する。

再配置が発生する。

[引数]

```

long    tEHdl    テキストエディットレコード
                へのハンドル
long    xpoint   水平座標(ローカル座標)
long    ypoint   垂直座標(ローカル座標)
long    columnPtr 段落情報を作成するバッファ
                ($18 バイト)のアドレス

```

[返り値]

```

DO.L    この段落のバイト位置/リザルトコード
AO.L    段落情報を作成するバッファへのアドレス

```

[コール]

```

pea      columnPtr
move.l   # ypoint,-(sp)
move.l   # xpoint,-(sp)
pea      tEHdl
SXCALL   $A341
lea      16(sp),sp

```

\$A343 TmCalSelPoint

tEHdlで指定されたテキストエディットの現在のセレクト領域の行位置と座標を再計算する。

再配置が発生する。

[引数]

```

long    tEHdl    テキストエディットレコード
                へのハンドル

```

[返り値]

```

DO.L    リザルトコード

```

[コール]

```

pea      tEHdl
SXCALL   $A343
addq.l   # 4,sp

```

\$A345 TMSetView

tEHdlで指定したテキストエディットのビューレクタングルとして、viewRectで指定したレクタングルを設定する。

再配置が発生する。

[引数]

```

long    tEHdl    テキストエディットレコード
                へのハンドル
long    viewRect レクタングルレコードのアド
                レス

```

[返り値]

```

DO.L    リザルトコード

```

[コール]

```

pea      viewRect
pea      tEHdl
SXCALL   $A345
addq.l   # 8,sp

```

\$A346 TMScroll

tEHdlで指定したテキストエディットを、dh, dvで指定したドットだけ縦・横にスクロールさせ、再表示する。dhは正で右、負で左方向へ、dvは正で下、負で上方向にスクロールする。

再配置が発生する。

[引数]

```

long    tEHdl    テキストエディットレコード
                へのハンドル
long    dh        水平方向にスクロールさせる
                ドット数
long    dv        垂直方向にスクロールさせる
                ドット数

```

[返り値]

```

DO.L    リザルトコード

```

[コール]

```

move.l   # dv,-(sp)
move.l   # dh,-(sp)
pea      tEHdl
SXCALL   $A346
lea      12(sp),sp

```

\$A347 TmPointScroll

tEHdlで指定したテキストエディットのビューレクタングルに、xpoint, ypointで指定した座標が収まるようにスクロールさせ、再表示する。

再配置が発生する。

[引数]

long	tEHdl	テキストエディットレコードへのハンドル
long	xpoint	水平座標(ローカル座標)
long	ypoint	垂直座標(ローカル座標)

[返り値]

DO.L リザルトコード

[コール]

```
move.l    # ypoint, -(sp)
move.l    # xpoint, -(sp)
pea       tEHdl
SXCALL    $A347
lea       12(sp), sp
```

\$A348 TMStr2

tEHdlで指定されたテキストエディットレコードの編集テキストのセレクト領域を、textPtrで指定した文字列と置き換え、編集履歴レコードを作成する。カーソル位置の再計算は行われない。

再表示は行わない。

再配置が発生する。

[引数]

long	tEHdl	テキストエディットレコードへのハンドル
long	textPtr	テキストへのポインタ
long	length	テキストのバイト数
long	hisRecPtr	編集履歴レコードのアドレス

[返り値]

DO.L リザルトコード
 =0 編集しなかった
 =1 編集した

[コール]

```
pea       hisRecPtr
move.l    # length, -(sp)
pea       textPtr
pea       tEHdl
SXCALL    $A348
lea       16(sp), sp
```

\$A349 TMKeyToAsk

eventRecで指定したイベントレコードに格納されているキーコードを、tEHdlで指定したテキストエディットレコードのファンクションキーアサインテーブルによって変換し、イベントレコードに再設定する。

[引数]

long	tEHdl	テキストエディットレコードへのハンドル
long	eventRec	イベントレコードのアドレス

[返り値]

DO.L リザルトコード
 =-1 キーコードが格納されていない
 =-2 該当するコードがアサインテーブルに登録されていない

[コール]

```
pea       eventRec
pea       tEHdl
SXCALL    $A349
addq.l    # 8, sp
```

\$A34A TMNextCode

次のキーダウンイベントを先読みし、tEHdlで指定したテキストエディットレコードのキーアサインテーブルによってキーコードを変換した後、codeで指定したキーコードであるかどうかチェックする。指定したコードであった場合、modeに0以外を指定すると、このイベントを取り除く。

[引数]

long	tEHdl	テキストエディットレコードへのハンドル
word	code	キーコード =0 すべてのコード
word	mode	=0 イベントを取り除かない ≠0 イベントを取り除く

[返り値]

DO.L 上位ワード: キーコード
 下位ワード: ASCII コード
 =0 指定したコードのキーイベントは発見できなかった

[コール]

```
move.w    # mode, -(sp)
move.w    # code, -(sp)
pea       tEHdl
SXCALL    $A34A
addq.l    # 8, sp
```


\$A34B TMSetTextH

tEHdlで指定したテキストエディットに、textHdlとlengthで指定したテキストを編集用にセットする。再表示は行わない。

疑似ハンドルは不可。

再配置が発生する。

[引数]

long	tEHdl	テキストエディットレコードへのハンドル
long	textHdl	テキストへのハンドル
long	length	テキストのバイト数

[返り値]

DO.L リザルトコード

[コール]

```

move.l    # length, -(sp)
pea       textHdl
pea       tEHdl
SXCALL    $A34B
lea       12(sp), sp

```

\$A460 TMNextCodeIn

次のキーダウンイベントを先読みし、tEHdlで指定したテキストエディットレコードのキーアサインテーブルによってキーコードを変換した後、codeで指定したキーコードであるかどうかチェックする。指定したコードを含むイベントが発生するまで、すべてのキーダウンイベントを取り除く。

[引数]

long	tEHdl	テキストエディットレコードへのハンドル
word	code	キーコード =0 すべてのコード

[返り値]

DO.L 上位ワード：キーコード
下位ワード：ASCIIコード
=0 指定したコードのキーイベントは見えなかった

[コール]

```

move.w    # code, -(sp)
pea       tEHdl
SXCALL    $A460
addq.l    # 6, sp

```

\$A462 TMSelReverse

tEHdlで指定したテキストエディットのselStart, selEndで指定した範囲を反転表示する。ハイライト表示属性が負の場合は何もしない。

再配置が発生する。

[引数]

long	tEHdl	テキストエディットレコードへのハンドル
long	selStart	開始位置(バイト位置)
long	selEnd	終了位置(バイト位置)

[返り値]

DO.L リザルトコード

[コール]

```

move.l    # selEnd, -(sp)
move.l    # selStart, -(sp)
pea       tEHdl
SXCALL    $A462
lea       12(sp), sp

```

\$A463 TMTini

テキストマネージャの終了処理を行う。
再配置が発生する。

[引数]

なし

[返り値]

DO.L リザルトコード

\$A464 TMSetSelCal

tEHdlで指定されたテキストエディットレコードの編集テキストのセレクト領域を、selStart, selEndで指定した領域にあらたに設定し、全体を計算し直す。再表示は行わない。selOffsetは、前のセレクト位置の開始位置、あるいは終了位置を指定する。開始位置を変更する場合は前の開始位置を、終了位置を変更する場合は前の終了位置を指定する。

再配置が発生する。

[引数]

long	tEHdl	テキストエディットレコードへのハンドル
long	selStart	セレクト範囲の先頭位置のオフセット
long	selEnd	セレクト範囲の終端位置のオフセット

フセット

long selOffset 前のセレクト位置のオフセット

[返り値]

DO.L リザルトコード

[コール]

move.l #selOffset,-(sp)

move.l #selEnd,-(sp)

move.l #selStart,-(sp)

pea tEHdl

SXCALL \$A364

lea 16(sp),sp

\$A465 TMActivate?

tEHdlで指定したテキストエディットのハイライト表示レベルを+1して、カーソルを表示、あるいはセレクト領域を反転表示する。

再配置が発生する。

[引数]

long tEHdl テキストエディットレコードへのハンドル

[返り値]

DO.L リザルトコード

[コール]

pea tEHdl

SXCALL \$A465

addq.l #4,sp

\$A466 TMDeactivate?

tEHdlで指定したテキストエディットについて、カーソルを消去、あるいはセレクト領域を反転表示を消した後、ハイライト表示レベルを-1する。

再配置が発生する。

[引数]

long tEHdl テキストエディットレコードへのハンドル

[返り値]

DO.L リザルトコード

[コール]

pea tEHdl

SXCALL \$A466

addq.l #4,sp

\$A467 TMCheckSel

tEHdlで指定されたテキストエディットレコード

の編集テキストのセレクト領域をselStart, selEndで指定した領域にあらたに設定し、再表示を行う。selOffsetは、前のセレクト位置の開始位置、あるいは終了位置を指定する。開始位置を変更する場合は、前の開始位置を、終了位置を変更する場合は前の終了位置を指定する。カーソル位置の再計算は行わない。

再配置が発生する。

[引数]

long tEHdl テキストエディットレコードへのハンドル

long selStart セレクト範囲の先頭位置のオフセット

long selEnd セレクト範囲の終端位置のオフセット

long selOffset 前のセレクト位置のオフセット

[返り値]

DO.L リザルトコード

[コール]

move.l #selOffset,-(sp)

move.l #selEnd,-(sp)

move.l #selStart,-(sp)

pea tEHdl

SXCALL \$A467

lea 16(sp),sp

\$A468 TMCaIPoint?

tEHdlで指定したテキストエディットのoffsetで指定したバイト位置の座標を計算し、buffPtrに返す。

バッファに返される情報の形式は以下のとおり。

+\$00.L	水平座標
+\$04.L	垂直座標
+\$08.L	行揃えのための補正值

再配置が発生する。

[引数]

long tEHdl テキストエディットレコードへのハンドル

long offset バイト位置

long buffPtr 座標情報が返るバッファ(12バイト)のアドレス

[返り値]

A0.L 座標情報が返るバッファのアドレス

[コール]

```

pea      buffPtr
move.l   #offset,-(sp)
pea      tEHdl
SXCALL   $A468
lea      12(sp),sp

```

\$A46A TMISZen

tEHdlで指定したテキストエディットのoffsetで指定したバイト位置がシフト JIS コードの何バイト目かを調べる。

再配置が発生する。

[引数]

```

long    tEHdl    テキストエディットレコード
                     へのハンドル

```

```

long    offset    バイト位置

```

[返り値]

```

D0.L    リザルトコード
        =0      シフト JIS コードの1バイト目
        =1      シフト JIS コードの2バイト目

```

```

A0.L    指定された位置のテキストのアドレス

```

[コール]

```

move.l   #offset,-(sp)
pea      tEHdl
SXCALL   $A46A
addq.l   #8,sp

```

\$A46B TMSetDestOffset

tEHdlで指定したテキストエディットの、ビューレクタングルに対するディスティネーションレクタングルのオフセットとして offsetx, offsety をセットし、再表示する。

再配置が発生する。

[引数]

```

long    tEHdl    テキストエディットレコード
                     へのハンドル
long    offsetx   水平座標オフセット
long    offsety   垂直座標オフセット

```

[返り値]

```

D0.L    リザルトコード

```

[コール]

```

move.l   #offsety,-(sp)

```

```

move.l   #offsetx,-(sp)
pea      tEHdl
SXCALL   $A46B
lea      12(sp),sp

```

\$A46C TMGetDestOffset

tEHdlで指定したテキストエディットの、ビューレクタングルに対するディスティネーションレクタングルのオフセットを、buffPtrで指定したバッファに返す。

バッファに返される情報の形式は以下のとおり。

+\$00.L	水平座標オフセット
+\$04.L	垂直座標オフセット

[引数]

```

long    tEHdl    テキストエディットレコード
                     へのハンドル
long    buffPtr   結果が返るバッファ(8バイト)のアドレス

```

[返り値]

```

A0.L    結果が返るバッファのアドレス

```

[コール]

```

pea      buffPtr
pea      tEHdl
SXCALL   $A46C
addq.l   #8,sp

```

\$A46D TMGetSelect

tEHdlで指定したテキストエディットの現在のセレクト状態を buffPtr で指定したバッファに返す。

バッファに返される情報の形式は以下のとおり。

+\$00.L	セレクト開始位置
+\$04.L	セレクト終了位置
+\$08.L	現在のセレクト位置

[引数]

```

long    tEHdl    テキストエディットレコード
                     へのハンドル
long    buffPtr   結果が返るバッファ(12バイト)のアドレス

```

[返り値]

```

D0.L    リザルトコード

```

```

A0.L    結果が返るバッファのアドレス

```

[コール]

```

pea      buffPtr
pea      tEHdl
SXCALL   $A46D
addq.l   #8,sp

```

ライブラリ TMEventW

tEHdlで指定したテキストエディットに格納されているグラフポートレコードへのポインタ (teInPort) がウィンドウレコードへのポインタであると仮定し、アップデートリージョンの範囲内を再描画し、再描画した範囲はアップデートリージョンから取り除く。その後、\$A31C TMEventと同様な処理を行う。

アップデートリージョンを操作するので、\$A20D WMUpdate後には使用できない。

再配置が発生する。

[引数]

```

long    tEHdl      テキストエディットレコード
                        へのハンドル
long    eventRec    イベントレコードのアドレス

```

[返り値]

```

DO.L    リザルトコード
        =0      編集しなかった
        =1      編集した

```

[コール]

```

pea      eventRec
pea      tEHdl
jsr      _TMEventW
addq.l   #8,sp

```

ライブラリ TMUpdateExist

tEHdlで指定したテキストエディットに格納されているグラフポートレコードへのポインタ (teInPort) がウィンドウレコードへのポインタであると仮定し、アップデートリージョンの範囲内を再描画し、modeが0以外の場合は再描画した範囲をアップデートリージョンから取り除く。

アップデートリージョンを操作するので、\$A20D WMUpdate後には使用できない。

再配置が発生する。

[引数]

```

long    tEHdl      テキストエディットレコード
                        へのハンドル
long    mode        =0  描画した範囲をアップ
                        デートリージョンから
                        取り除かない
                        ≠0  描画した範囲をアップ
                        デートリージョンから
                        取り除く

```

[返り値]

```

DO.L    リザルトコード
        =0      アップデートリージョンが存
                        在しない
        =1      アップデートリージョンを描
                        画した

```

[コール]

```

move.l   #mode,-(sp)
pea      tEHdl
jsr      _TMUpdateExist
addq.l   #8,sp

```

タスクマネージャ

\$A414 ●

eventRecで指定したタスクマネージャイベントレコードの内容と、Hmodel、Hmode2を、taskIDで指定したタスクにイベントとして発生させる。

指定したタスクマネージャイベントレコードのタグ、whomあるいはwhom2に格納されている引数がハンドルであり、タスクマネージャイベントレコード

を廃棄する際、それらが指すブロックを同時に廃棄してもよい場合、HmodelあるいはHmode2に0以外を指定する。

再配置が発生する。

[引数]

```

long    tsEventRec  タスクマネージャイベントレ
                        コードのアドレス

```



```

byte   Hmodel   ハンドルモード 1
byte   Hmode2   ハンドルモード 2
word   taskID   タスク ID

```

[返り値]

```
DO.L   リザルトコード
```

[コール]

```

move.w   # taskID, -(sp)
move.w   # Hmodel*$100+Hmode2, -(sp)
*byte でスタック操作はできない
pea      eventRec
SXCALL   $A414
addq.l   # 8, sp

```

\$A415 TSPostEventTsk2

mes1, mes2, what2で指定したデータを、それぞれタグの whom, whom2, what2に持つようなタスクマネージャイベントレコードを作成し、taskIDで指定したタスクにイベントとして発生させる。このイベントの種類は 12 (E_SYSTEM1) に固定。

whomあるいはwhom2に格納されている引数がハンドルである場合、HmodelあるいはHmode2に0以外を指定すると、別のブロックを作成してwhom, whom2が指すブロックの内容をコピーする。タスクマネージャのイベントキューに登録されるタスクマネージャイベントレコードのタグwhom, whom2には新しく作成したブロックへのハンドルが格納される。この場合、タスクマネージャイベントレコードを廃棄する際、これらのブロックは同時に廃棄される。

再配置が発生する。

[引数]

```

long    mes1     メッセージ 1
long    mes2     メッセージ 2
word    what2    タスクマネージャイベントコード
byte    Hmodel   ハンドルモード 1
byte    Hmode2   ハンドルモード 2
word    taskID   イベントを発生させるタスクの ID

```

[返り値]

```
DO.L   リザルトコード
```

[コール]

```

move.w   # taskID, -(sp)
move.w   # Hmodel*$100+Hmode2, -(sp)
*byte でスタック操作はできない
move.w   # what2, -(sp)

```

```

pea      mes2
pea      mes1
SXCALL   $A415
lea      14(sp), sp

```

\$A416

\$A3F4 TSFindTsknの下請けルーチン。削除されたタスクやロードしただけのタスク、そして、終了後メモリから削除されていないタスクについても検索することができる。

[引数]

```

long    namePtr   タスク名を示す文字列(ASCIIZ)のアドレス
word    taskID    タスク ID

```

[返り値]

```
DO.L   タスクID(下位ワードのみ有効)/リザルトコード
```

[コール]

```

move.w   # taskID, -(sp)
pea      namePtr
SXCALL   $A416
addq.l   # 6, sp

```

\$A417 TSAnswer

tsEventRecで指定したイベントレコードの内容を、タスク間通信に対する返事として通信相手のタスクに返す。

再配置が発生する。

[引数]

```
long    tsEventRec タスクマネージャイベントレコードのアドレス
```

[返り値]

```

DO.L   =0      正常終了
       =-1     通信中ではない
       =-2     宛先のタスクの準備が整っていない

```

[コール]

```

pea      tsEventRec
SXCALL   $A417
addq.l   # 4, sp

```

\$A418 TSSendMes

listenerで指定したタスクに、tsEventRecで指定したタスクマネージャイベントレコードの内容を

メッセージとして送信する。返事があった場合は、tsEventRecの中に返事が返る。

再配置が発生する。

[引数]

word listener 宛先となるタスクの ID
long tsEventRec タスクマネージャイベントレコードのアドレス

[返り値]

DO.L =0 正常終了
=2 返事が届いた
=-1 宛先のタスクが存在しない/
通信中
=-2 宛先のタスクの準備が整っていない

[コール]

```
pea      tsEventRec
move.w   # listener, -(sp)
SXCALL   $A418
addq.l   # 6, sp
```

\$A419 TSGetMes

tsEventRecで指定したタスクマネージャイベントレコードにメッセージを読み込む。modeで0を指定した場合は、相手には-2（「宛先のタスクの準備が整っていない」）が、0以外を指定した場合は0（「正常終了」）が返る。

起動直後にメッセージを受けつける場合に使用する。

[引数]

long tsEventRec タスクマネージャイベントレコードのアドレス
word mode =0 受けつけなかったことにする
≠0 受けつけたことを通知する

[返り値]

DO.L =0 メッセージは届いていない
=1 メッセージを受け取った

[コール]

```
move.w   # mode, -(sp)
pea      tsEventRec
SXCALL   $A419
addq.l   # 6, sp
```

\$A41A TSInitTsk2

タスクマネージャ以下のマネージャを、すべて初期化する。\$A34C TSInitTskとの違いは、rscfileによってシステムリソースファイルを指定できる点。rscfileが0、あるいは先頭1バイトが0の場合、デフォルトのSYSTEM.LBが使用される。また、実際にオープンしたファイル名がrscfileに格納される。

[引数]

long memStart ヒープゾーン先頭アドレス
long memEnd ヒープゾーン終了アドレス
long path カレントパスを示す文字列のアドレス(ASCIIZ)
byte model シェルのリリース番号(1~)
byte mode2 システムリソースヒープゾーン作成フラグ
=0 作成しない
≠0 作成する
word ver シェルのバージョン
long rscfile システムリソース名(ASIIZ)
(ファイル名が返るので90バイト必要)

[返り値]

DO.L ヒープゾーンのアドレス/リザルトコード
AO.L システムリソースへのハンドル

[コール]

```
pea      rscfile
move.w   # ver, -(sp)
move.w   # model*$100+mode2, -(sp)
*byte でスタック操作はできない
pea      path
pea      memEnd
pea      memStart
SXCALL   $A41A
lea      20(sp), sp
```

\$A41F SXCallWindM2

\$A3A2 SXCallWindMと同様な処理を行う。rectPtrで指定するレクタングルでウィンドウサイズ変更時の最大サイズ、最小サイズを指定できる点が異なる。左上の座標が最小サイズ、右下の座標が最大サイズを意味する。

再配置が発生する。

[引数]

long	winPtr	ウィンドウレコードのアドレス
long	tsEventRec	タスクマネージャイベントレコードのアドレス
long	rectPtr	レクタングルレコードのアドレス

[返り値]

DO.L	ウィンドウのパートコード/リザルトコード
------	----------------------

[コール]

pea	rectPtr
pea	tsEventRec
pea	winPtr
SXCALL	\$A41F
lea	l2(sp),sp

\$A420 TSBEGINDRAG2

pt で指定したポイントからドラッグを開始する。\$A38A TSBEGINDRAG2 とは、ラバーバンド表示ルーチンを指定できる点で異なる。procPtr で 0 を指定すると、標準のラバーバンド表示ルーチンが使用されるが、この場合はアイコン管理レコードのみサポートされる。

あらかじめグラフポートをセットしておく必要がある。

ラバーバンド表示ルーチンへは、次のようなパラメータが AO 経由で渡される。

(a0). L	ドラッグレコードへのポインタ
4 (a0). W	コマンド
6 (a0). l	TSBEGINDRAG2 で指定したパラメータ

コマンドには「初期化」、「表示」、「消去」、「終了」の 4 つが存在し、その仕様は以下のとおり。

・command=0: 初期化

最初に一度だけ呼ばれる。標準のラバーバンド表示ルーチンではビットイメージの作成が行われる。

・command=1: 終了

終了時に呼ばれる。初期化時に確保したメモリの廃棄等を行う。

・command=2: 表示

ラバーバンドを表示する。ドラッグレコードの drOrigin を引数にして GMSetHome をコールすることで、移動した分のローカル座標がセットされる。

・command=3: 消去

ラバーバンドを消去する。ドラッグレコードの drOrigin を引数にして GMSetHome をコールする

ことで、移動した分のローカル座標がセットされる。

[引数]

long	pt	ドラッグ開始位置(グローバル座標)
long	procPtr	ラバーバンド表示ルーチンのアドレス
long	param	ラバーバンド表示ルーチンに渡されるパラメータ

[返り値]

DO.L	リザルトコード
------	---------

[コール]

move.l	# param,-(sp)
pea	procPtr
move.l	# pt,-(sp)
SXCALL	\$A420
lea	l2(sp),sp

\$A421

strZPtr で指定された文字列の中からパス名で 사용되는キャラクタ (' ¥ ' , ' / ' , ' . ') を探し出す。

[引数]

long	strZPtr	文字列(ASCIIZ)のアドレス
------	---------	------------------

[返り値]

DO.L	=0	発見できた
	=1	発見できなかった
AO.L		発見したアドレス

[コール]

pea	strZPtr
SXCALL	\$A421
addq.l	# 4,sp

\$A422 SXGETVECTOR

intNo で指定した SX コールのベクタを返す。

[引数]

word	intNo	SX コール番号
------	-------	----------

[返り値]

DO.L	ベクタの内容/リザルトコード
AO.L	ベクタの内容

[コール]

move.w	# intNo,-(sp)
SXCALL	\$A422
addq.l	# 2,sp

\$A423 SXSetVector

intNoで指定したSXコールのベクタとしてvectorを設定する。

[引数]

word	intNo	SX コール番号
long	vector	登録するアドレス

[返り値]

D0.L	ベクタの内容/リザルトコード
A0.L	ベクタの内容

[コール]

```

pea    vector
move.w #intNo, -(sp)
SXCALL $A423
addq.l #6, sp

```

\$A424 ●

ISRecで指定したアイコン管理レコードで描画されるアイコンの、ファイル名を表示すべき領域を示すレクタングルをbuffPtrに返す。描画すべきアイコンのイメージを収めたリソースが読み込まれ、そのリソースタイプがD0に、ハンドルがA0に返る。

再配置が発生する。

[引数]

long	ISRec	アイコン管理レコードのアドレス
long	buffPtr	レクタングルが返るバッファ (8 バイト)

[返り値]

D0.L	アイコンのイメージが収められているリソースタイプ
A0.L	アイコンのイメージが収められているメモリ上のリソースへのハンドル

[コール]

```

pea    buffPtr
pea    ISRec
SXCALL $A424
addq.l #8, sp

```

\$A425 ●

メモリ中に読み込まれたXタイプの実行ファイルをリロケートし、BSSセクションをクリアする。

[引数]

long	codePtr	コード部の先頭アドレス
long	headPtr	Xタイプ実行ファイルヘッダ先頭アドレス
long	ofsPtr	オフセットテーブル先頭アドレス

[返り値]

D0.L	リザルトコード
A0.L	BSS セクション末尾+1

[コール]

```

pea    ofsPtr
pea    headPtr
pea    codePtr
SXCALL $A425
lea    12(sp), sp

```

\$A426 ●

blk1Ptr, blk1Lenで示されるブロックの中のinsOfsで示される場所に、blk2Ptr, blk2Lenで示されるブロックを挿入する。その際、insOfsにもともとあった部分をdstOfsで示される場所に転送する。

[引数]

long	blk1Ptr	メモリブロックへのポインタ
long	blk1Len	メモリブロックのサイズ
long	insOfs	ブロック2を挿入するオフセット
long	dstOfs	挿入する部分を転送するオフセット
long	blk2Ptr	挿入するブロックへのポインタ
long	blk2Len	挿入するブロックのサイズ

[返り値]

D0.L	ブロック転送したバイト数
------	--------------

[コール]

```

move.l #blk2Len, -(sp)
pea    blk2Ptr
move.l #dstOfs, -(sp)
move.l #insOfs, -(sp)
move.l #blk1Len, -(sp)
pea    blk1Ptr
SXCALL $A426
lea    24(sp), sp

```

\$A427 TSCellToStr

celHdlで指定したセルリストから文字列を抽出し、buffPtrで指定したバッファに返す。maxで指

定したバッファのサイズを超えた場合、エラーとなる。
buffPtr として 0 を指定した場合、ヒープ上に再配置可能ブロックを作成して、そこに収めてハンドルを AO に返す。

再配置が発生する。

[引数]

long	celHdl	セルリストへのハンドル
long	buffPtr	結果が返るバッファのアドレス
long	max	バッファのサイズ

[返り値]

DO.L	文字列のバイト数/リザルトコード
AO.L	文字列を収めたバッファへのハンドル(確保しなかった場合は 0)

[コール]

```

move.l    # max, -(sp)
pea       buffPtr
pea       celHdl
SXCALL    $A427
lea       12(sp), sp

```

\$A428 ●

celHdl で指定したセルリストから文字列を抽出し、buffPtr で指定したバッファに返す。max で指定したバッファのサイズを超えた場合、エラーとなる。buffPtr として 0 を指定した場合、ヒープ上に再配置可能ブロックを作成して、そこに収めてハンドルを AO に返す。セルリストにアイコン管理レコードが含まれている場合はフルパスのファイル名を抽出する。

再配置が発生する。

[引数]

long	celHdl	セルリストへのハンドル
long	buffPtr	結果が返るバッファのアドレス
long	max	バッファのサイズ

[返り値]

DO.L	文字列のバイト数/リザルトコード
AO.L	文字列を収めたバッファへのハンドル(確保しなかった場合は 0)

[コール]

```

move.l    # max, -(sp)
pea       buffPtr
pea       celHdl
SXCALL    $A428
lea       12(sp), sp

```

\$A429 ●

ShMEHdl で指定したシェル用メニューテンプレートの、itemNo で指定するアイテムの表示文字列を itemNamePtr に、実行ファイル名を fileNamePtr に返す。また、機能番号が設定されている場合は DO に返る。

[引数]

long	ShMEHdl	シェル用メニューテンプレートへのハンドル
word	itemNo	アイテム番号
long	itemNamePtr	表示文字列(ASCIIIZ)が返るバッファのアドレス
long	fileNamePtr	実行ファイル名(ASCIIIZ)が返るバッファのアドレス

[返り値]

DO.L	=0	ファイル名等をコピーした
	≠0	機能番号

[コール]

```

pea       fileNamePtr
pea       itemNamePtr
move.w    # itemNo, -(sp)
pea       ShMEHdl
SXCALL    $A429
lea       14(sp), sp

```

\$A42A SXLockFSX

SX-SYSTEM をロックする。

[引数]

なし

[返り値]

なし

\$A42B SXUnlockFSX

SX-SYSTEM をアンロックする。

[引数]

なし

[返り値]

なし

\$A42C TSFockMode

fileNameで指定したファイルを\$A353 FockBで起動した場合の起動モードを調べる（起動するわけではない）。dFileNameとfileNameは、同じものを指定してもよい。

再配置が発生する。

[引数]

long	fileName	ファイル名(ASCIIIZ)のアドレス
long	dFileName	実際に起動されるファイル名が返るバッファ(90バイト)

[返り値]

DO.L	=0	ファイルから起動
	=1	リソースタイプ CODE から起動
	=2	メモリ中のタスクを複写して起動
AO.L	複写するタスクの ID/リソース CODE の ID	

[コール]

```

pea    dFileName
pea    fileName
SXCALL $A42C
addq.l #8,sp

```

\$A42D ●

fileNameで指定した名前と同じ名前を持つタスクが存在するかどうか調べる。

[引数]

long	fileName	ファイルネーム(ASCIIIZ)のアドレス
------	----------	-----------------------

[返り値]

DO.L	上位ワード：現在のタスクの状態 下位ワード：タスク ID/リザルトコード
AO.L	タスク管理テーブルのアドレス

[コール]

```

pea    fileName
SXCALL $A42D
addq.l #4,sp

```

\$A42E ●

ofsPtで指定した値を画面のスクロールレジスタ(テキスト, グラフィック 0~3 ページ)に代入し、ハードウェアスクロールを行う。

[引数]

long	ofsPt	水平・垂直にスクロールするオフセットを意味するポイント
------	-------	-----------------------------

[返り値]

なし

[コール]

```

move.l    #ofsPt, -(sp)
SXCALL    $A42E
addq.l    #4, sp

```

\$A42F ●

現在のスクロールレジスタの値を返す。ただし、スクロールレジスタは読み出しができないので、SX-SYSTEM 内部に保存してある、前回設定した値を読み出している。

[引数]

なし

[返り値]

DO.L	水平・垂直にスクロールするオフセットを意味するポイント
------	-----------------------------

\$A430 TSSetGraphMode

タスクマネージャを初期化する際に参照される画面モードを SRAM に設定する。

scrModeは IOCS \$10 _CRTMODで指定する値と同等。rmodeとして0以外を指定すると、実画面モードとなる。

[引数]

word	rmode	0 以外で実画面モード
word	scrMode	画面モード

[返り値]

DO.L	リザルトコード
------	---------

[コール]

```

move.w    #scrMode, -(sp)
move.w    #rmode, -(sp)
SXCALL    $A430
addq.l    #4, sp

```

\$A431 TSGetGraphMode

タスクマネージャの画面モードを得る。

[引数]

なし

[返り値]

DO.L SRAM に設定されている画面モード
上位ワード：0 以外で実画面モード
下位ワード：画面モード

AO.L グラフィックマネージャを初期化した値
上位ワード：0 以外で実画面モード
下位ワード：画面モード
=-1 グラフィックマネージャは初期化されていない

\$A432 SXGetDispRect

rectPtr で指定したレクタングルレコードに現在の表示画面を返す。実画面モードの場合、表示画面と実際に描画を行う実画面は異なる。

[引数]

long rectPtr レクタングルレコードのアドレス

[返り値]

なし

[コール]

```

    pea    rectPtr
    SXCALL $A432
    addq.l #4,sp

```

\$A433 ●

taskID で指定したタスク ID を持ち、tick で指定したシステム時刻に起動されたタスクの状態を返す。

[引数]

long tick システム時刻
word taskID タスク ID

[返り値]

DO.L タスクの状態
=-1 エラー

[コール]

```

    move.w #taskID,-(sp)
    move.l #tick,-(sp)
    SXCALL $A433
    addq.l #6,sp

```

\$A434 ●

ISRec で指定したアイコン管理レコードの内容にしたがってアイコンを描画する。rectPtr で指定したレクタングルレコードに、ファイル名を表示すべき領域を示すレクタングルを返す。

再配置が発生する。

[引数]

long ISRec アイコン管理レコードのアドレス
long rectPtr レクタングルレコードのアドレス

[返り値]

DO.L リザルトコード

[コール]

```

    pea    rectPtr
    pea    ISRec
    SXCALL $A434
    addq.l #8,sp

```

\$A435 SXSRAMVer

SRAM を初期化した SX-SYSTEM のバージョンを得る。

[引数]

なし

[返り値]

DO.L バージョン(SX1.10 の場合は 1,それ以前は 0)

\$A436 SXSRAMReset

SRAM の、SX-SYSTEM が使用する領域を初期化する。

[引数]

なし

[返り値]

なし

\$A437 SXSRAMCheck

SRAM に記録されている情報のバージョンを調べ、自分より古い場合は初期化を行う。

[引数]

なし

[返り値]

DO.L	=0	自分と同じバージョンなので初期化を行わなかった
	=1	初期化を行った
	=2	自分より新しいバージョンなので初期化を行わなかった

ライブラリ TSSetAbort

ハードウェアエラーが発生した場合に実行するアボート処理ルーチンを設定する。エラーが発生した場合、exit () で終了する前にアボート処理ルーチンが (*procPtr) (-8194, param) のかたちで呼び出される。

Cでコンパイルしたプログラムでのみ有効。

[引数]

long	procPtr	アボート処理ルーチンのアドレス
long	param	アボート処理ルーチンに渡されるパラメータ

[返り値]

DO.L	前の処理ルーチンのアドレス
------	---------------

[コール]

```
move.l    # param, -(sp)
pea       procPtr
jsr       _TSSetAbort
addq.l    # 8, sp
```

グラフィックマネージャ

\$A16C GMImgToRgn

bitmapPtrで指定されたビットマップの rectPtr で指定された範囲内で、カラーコードが0以外の領域を求め、rgnHdlで指定したリージョンに収める。ビットマップがテキストタイプの場合はアクセスページが参照される。

再配置が発生する。

[引数]

long	rgnHdl	リージョンレコードへのハンドル
long	bitmapPtr	ビットマップレコードのアドレス
long	rectPtr	レクタングルレコードのアドレス

[返り値]

DO.L	リザルトコード
AO.L	リージョンレコードへのハンドル

[コール]

```
pea       rectPtr
pea       bitmapPtr
pea       rgnHdl
SXCALL    $A16C
lea       l2(sp), sp
```

\$A178 GMFrameArc

rectPtr, startAngle, endAngleで指定された円弧の枠を描画する。

スクリプトに記録される。

リージョンに記録される。

[引数]

long	rectPtr	円弧が内接するレクタングルレコードのアドレス
word	startAngle	開始角度
word	endAngle	終了角度

[返り値]

DO.L	リザルトコード
------	---------

[コール]

```
move.w    # endAngle, -(sp)
move.w    # startAngle, -(sp)
pea       rectPtr
SXCALL    $A178
addq.l    # 8, sp
```

\$A179 GMFillArc

rectPtr, startAngle, endAngleで指定された円弧の内部を塗り潰す。

スクリプトに記録される。

リージョンに記録される。

[引数]

long rectPtr 円弧が内接するレクタングル
レコードのアドレス

word startAngle 開始角度

word endAngle 終了角度

[返り値]

DO.L リザルトコード

[コール]

```
move.w    #endAngle, -(sp)
move.w    #startAngle, -(sp)
pea       rectPtr
SXCALL    $A179
addq.l    #8, sp
```

\$A17C GMFramePoly

polyHdlで指定したポリゴンの枠を描画する。
スクリプトに記録される。
リージョンに記録される。

[引数]

long polyHdl ポリゴンレコードへのハンドル

[返り値]

DO.L リザルトコード

[コール]

```
pea       polyHdl
SXCALL    $A17C
addq.l    #4, sp
```

\$A17D GMFillPoly

polyHdlで指定したポリゴンの内部を塗り潰す。
スクリプトに記録される。
リージョンに記録される。

[引数]

long polyHdl ポリゴンレコードへのハンドル

[返り値]

DO.L リザルトコード

[コール]

```
pea       polyHdl
SXCALL    $A17D
addq.l    #4, sp
```

\$A19E GMOpenPoly

ポリゴンの記録を開始する。ドローレベルが-1され、以降実行されるGMLineで描画した座標が記録される。

再配置が発生する。

[引数]

なし

[返り値]

DO.L リザルトコード

\$A19F GMClosePoly

ポリゴンの記録を終了し、polyHdlで指定したポリゴンレコードに結果を返す。polyHdlとして0を指定すると、グラフィックマネージャがヒープ上に再配置可能ブロックを作成し、そこに結果を収めてハンドルを返す。

再配置が発生する。

[引数]

long polyHdl ポリゴンレコードへのハンドル

[返り値]

DO.L リザルトコード

AO.L ポリゴンレコードへのハンドル

[コール]

```
pea       polyHdl
SXCALL    $A19F
addq.l    #4, sp
```

\$A1A0 GMDisposePoly

polyHdlで指定したポリゴンレコードを廃棄する。

[引数]

long polyHdl ポリゴンレコードへのハンドル

[返り値]

DO.L リザルトコード

[コール]

```
pea       polyHdl
SXCALL    $A1A0
addq.l    #4, sp
```

\$A1A8 GMapPoly

srcRectPtrで指定されたレクタングル上にあるpolyHdlで指定したポリゴン、destRectPtr上に写像した結果をpolyHdl内に返す。どちらかのレクタングルがヌルレクタングルの場合、結果はヌルポリゴンとなる。

再配置が発生する。

[引数]


```

long    polyHdl    ポリゴンレコードへのハンドル
long    srcRectPtr  写像元のレクタングル
long    destRectPtr 写像先のレクタングル
[返り値]
DO.L    0
AO.L    ポリゴンレコードへのハンドル
[コール]
        pea        destRectPtr
        pea        srcRectPtr
        pea        polyHdl
        SXCALL     $A1A8
        lea        12(sp),sp

```

\$A1B9 GMCopyStdProc

gProcTblで指定したバッファに標準描画エントリテーブルを作成する。

```

[引数]
long    gProcTbl  描画エントリテーブルを返す
                        バッファ($40バイト)のアド
                        レス
[返り値]
DO.L    0
AO.L    結果が返るバッファのアドレス
[コール]
        pea        gProcTbl
        SXCALL     $A1B9
        addq.l     #4,sp

```

\$A1BB GMTransImg

異なるタイプのスクリーン間 (srcBitmapPtrで指定されるビットマップから destBitmapPtrで指定されるビットマップ) で、ビットイメージのコピーを行う。コピー元のビットマップ上の srcRectPtrで示されるレクタングル内部を、コピー先のビットマップ上の destRectPtrで示されるレクタングル内部にコピーするが、両レクタングルのサイズが異なる場合、拡大縮小が行われる。

コピー先のビットマップがカレントの場合、グラフポートレクタングル、ビズブルリージョンでクリッピングが行われる。

再配置が発生する。

```

[引数]
long    srcBitmapPtr
                        コピー元のビットマップレ

```

```

                        コードのアドレス
long    destBitmapPtr
                        コピー先のビットマップレ
                        コードのアドレス
long    srcRectPtr    コピー元のレクタングルレ
                        コードのアドレス
long    destRectPtr   コピー先のレクタングルレ
                        コードのアドレス

```

```

[返り値]
DO.L    リザルトコード
[コール]

```

```

        pea        destRectPtr
        pea        srcRectPtr
        pea        destBitmapPtr
        pea        srcBitmapPtr
        SXCALL     $A1BB
        lea        16(sp),sp

```

\$A1BC GMFillRimg

rectImgPtrで指定されるテキストタイプ・1ページのレクタングルイメージを、ペンモードにしたがって描画する。描画位置は pt で指定する。

再配置が発生する。

```

[引数]
long    rectImgPtr  レクタングルイメージレコー
                        ドのアドレス
long    pt          描画時に左上となるポイント
[返り値]
DO.L    リザルトコード
[コール]

```

```

        move.l     #pt, -(sp)
        pea        rectImgPtr
        SXCALL     $A1BC
        addq.l     #8,sp

```

\$A1BD GMFillImg

imgPtrで指定されるテキストタイプ・1ページのイメージを、ペンモードにしたがって描画する。描画位置、範囲は rectPtr で指定する。

再配置が発生する。

```

[引数]
long    imgPtr      イメージレコードのアドレス
long    rectPtr     描画位置、範囲を示すレクタ

```

シングルレコードのアドレス

[返り値]

DO.L リザルトコード

[コール]

```

pea    rectPtr
pea    imgPtr
SXCALL $A1BD
addq.l #8,sp

```

\$A1BE GMSlidedRgn

srcRgnHdlで指定されるリージョンを、ptで指定される移動量だけ相対移動し、その軌跡をdestRgnHdlにリージョンとして返す。

再配置が発生する。

[引数]

```

long    destRgnHdl 結果が返るリージョンレコードへのハンドル
long    srcRgnHdl  移動するリージョンレコードへのハンドル
long    pt          移動量を意味するポイント

```

[返り値]

DO.L リザルトコード

[コール]

```

move.l #pt, -(sp)
pea    rscRgnHdl
pea    destRgnHdl
SXCALL $A1BE
lea    12(sp), sp

```

\$A1BF GMPaintRgn

bitmapPtrで指定したビットマップのptで指定したポイントの周辺の同じ色の領域を、rgnHdlで指定されたリージョンに返す。

再配置が発生する。

[引数]

```

long    rgnHdl  結果が返るリージョンレコードへのハンドル
long    bitmapPtr ビットマップレコードのアドレス
long    pt      走査を開始するポイント

```

[返り値]

DO.L リザルトコード

AO.L 結果が返るリージョンレコードへのハンドル

[コール]

```

move.l #pt, -(sp)
pea    bitmapPtr
pea    rgnHdl
SXCALL $A1BF
lea    12(sp), sp

```

\$A1C0 GMSetRgnLine

リージョンデータの1行の最大バイト数をlengthに設定する。lengthは最大\$200までの偶数で、0を指定すると、初期状態に戻る。

[引数]

```

word    length  リージョンデータの1行の最大値

```

[返り値]

DO.L 前の最大値

[コール]

```

move.l #length, -(sp)
SXCALL $A1C0
addq.l #4, sp

```

\$A1C1 GMGetRgnLine

リージョンデータの1行の最大バイト数を返す。

[引数]

なし

[返り値]

DO.L 現在設定されている1行の最大バイト数

\$A1C2 GMInitGraphMode

scrModeで指定した画面モードにあわせてビットマップやグラフポートの初期値を設定する。scrModeで指定するのは、IOCS \$10 _CRTMODで指定する値と同等。

[引数]

```

word    scrMode  画面モード

```

[返り値]

DO.L リザルトコード

[コール]

```

move.w #scrMode, -(sp)
SXCALL $A1C2
addq.l #2, sp

```

\$A1C3 GMCurFont

カレントグラフィポートとフォントレコードを比較して状態が変化している場合は、フォントレコードをつくり直す。

再配置が発生する。

[引数]

なし

[返り値]

DO.L リザルトコード

AO.L フォントレコードへのハンドル

\$A1C4 GMGetScrnSize

現在設定されている画面のサイズを返す。

[引数]

なし

[返り値]

DO.L 画面の縦横のサイズを意味するポイント

\$A1C5 GMExgGraph

\$A131 GMSetGraphと同様に、graphPtrで指定するグラフィポートをカレントにする。DOにそれまでのカレントグラフィポートのアドレスが返る。

[引数]

long graphPtr グラフィポートレコードのアドレス

[返り値]

DO.L 0

AO.L 前のカレントグラフィポートのアドレス

[コール]

pea graphPtr

SXCALL \$A1C5

addq.l #4,sp

\$A1C6 GMExgBitmap

カレントグラフィポートにbitmapPtrで指定したビットマップレコードをセットする。AOにそれまでのビットマップレコードのアドレスが返る。

[引数]

long bitmapPtr ビットマップレコードのアドレス

[返り値]

DO.L 0

AO.L 前のビットマップレコードのアドレス

[コール]

pea bitmapPtr

SXCALL \$A1C6

addq.l #4,sp

\$A1C7 GMGetBitmap

カレントグラフィポートにセットされているビットマップレコードのアドレスを返す。

[引数]

なし

[返り値]

DO.L 0

AO.L ビットマップレコードのアドレス

\$A1C8 GMCalcBitmap

bitmapPtrで指定したビットマップレコードの内容を再計算する。具体的には、bmKind, bmRectをもとにして、line, pageを計算する。

[引数]

long bitmapPtr ビットマップレコードのアドレス

[返り値]

DO.L リザルトコード

AO.L ビットマップレコードのアドレス

[コール]

pea bitmapPtr

SXCALL \$A1C8

addq.l #4,sp

\$A1C9 GMCalcScrnSize

bitmapPtrで指定したビットマップが必要とするメモリのバイト数を計算する。

[引数]

long bitmapPtr ビットマップレコードのアドレス

[返り値]

DO.L ビットマップが必要とするバイト数

[コール]

pea bitmapPtr

SXCALL \$A1C9

addq.l #4,sp

\$A1CA GMNewBits

rectPtr で示した大きさを持つ、scrKind のタイプのビットを作成する。ビット内のビットマップレコードは、base を除いて初期化される。ビット内のイメージデータは初期化されない。

再配置が発生する。

[引数]

word	scrKind	スクリーンタイプ
long	rectPtr	ビットの大きさを示すレクタングルレコードのアドレス
word	aPage	アクセスページ(テキストタイプの場合)

[返り値]

D0.L	リザルトコード
A0.L	ビットへのハンドル

[コール]

```
move.w    # aPage,-(sp)
pea       rectPtr
move.w    # scrKind,-(sp)
SXCALL    $A1CA
addq.l    # 8,sp
```

\$A1CB GMDisposeBits

bitsHdl で指定したビットを廃棄する。

[引数]

long	bitsHdl	ビットへのハンドル
------	---------	-----------

[返り値]

D0.L	0
------	---

[コール]

```
pea       bitsHdl
SXCALL    $A1CB
addq.l    # 4,sp
```

\$A1CC GMLockBits

bitsHdl で指定したビットをロックし、ロックレベルを-1 する。

[引数]

long	bitsHdl	ビットへのハンドル
------	---------	-----------

[返り値]

D0.L	リザルトコード
A0.L	ビットへのハンドル

[コール]

```
pea       bitsHdl
SXCALL    $A1CC
addq.l    # 4,sp
```

\$A1CD GMUnlockBits

bitsHdl で指定したビットをアンロックし、ロックレベルを+1 する。

[引数]

long	bitsHdl	ビットへのハンドル
------	---------	-----------

[返り値]

D0.L	リザルトコード
A0.L	ビットへのハンドル

[コール]

```
pea       bitsHdl
SXCALL    $A1CD
addq.l    # 4,sp
```

\$A1CE GMItalicRect

rectPtr で指定したレクタングルを、イタリックの文字を傾けると同じだけ傾け、カレントグラフィポートに描画する。描画モードはフォントモードではなく、ペンモードが参照される。

[引数]

long	rectPtr	レクタングルレコードのアドレス
------	---------	-----------------

[返り値]

D0.L	リザルトコード
------	---------

[コール]

```
pea       rectPtr
SXCALL    $A1CE
addq.l    # 4,sp
```

\$A1CF GMItalicRgn

rectPtr で指定したレクタングルを、イタリックの文字を傾けると同じだけ傾けた領域を rgnHdl で指定されたリージョンレコードに返す。

再配置が発生する。

[引数]

long	rgnHdl	リージョンレコードへのハンドル
long	rectPtr	レクタングルレコードのアドレス

[返り値]

DO.L リザルトコード
AO.L リージョンレコードへのハンドル

[コール]

```
pea      rectPtr
pea      rgnHdl
SXCALL   $A1CF
addq.l   #8,sp
```

\$A1D0 GMFreeBits

bitsHdlで指定したビットのロックレベルを強制的に0にしてアンロックする。

[引数]

なし

[返り値]

DO.L 0
AO.L ビッツへのハンドル

\$A1D1 GMCalcGraph

graphPtrで指定したグラフポートにセットされているビットマップの情報をもとに、グラフポートレコードの内容を再計算する。ビジブルリージョン、クリップリージョン等が作成されていない場合、作成する。

再配置が発生する。

[引数]

long graphPtr グラフポートレコードのアドレス

[返り値]

DO.L リザルトコード
AO.L グラフポートレコードのアドレス

[コール]

```
pea      graphPtr
SXCALL   $A1D1
addq.l   #4,sp
```

\$A1D2 GMPackImage

srcPtr, srcLenで与えたメモリの内容をランレングス法で圧縮し、destPtrで指定したバッファに格納する。バイト単位で同じデータが3バイト以上連続した場合に圧縮を行うので、圧縮の結果得られるデータのサイズは、最悪の場合 $\text{srcLen} + ((\text{srcLen} + 127) / 127)$ バイトとなる。

[引数]

long destPtr 結果が収められるバッファのアドレス

long srcPtr 元のデータの先頭アドレス

long srcLen 元のデータのバイト数

[返り値]

DO.L 圧縮結果のバイト数/リザルトコード

AO.L 結果が収められるバッファの終端+1

[コール]

```
move.l   #srcLen, -(sp)
pea      srcPtr
pea      destPtr
SXCALL   $A1D2
lea      12(sp), sp
```

\$A1D3 GMUnpackImage

srcPtr, srcLenで与えた圧縮データを、destPtrで指定されたバッファに展開する

[引数]

long destPtr 結果が収められるバッファのアドレス

long srcPtr 元のデータの先頭アドレス

long srcLen 元のデータのバイト数

[返り値]

DO.L 展開前のバイト数/リザルトコード

AO.L 展開前のバッファの終端+1

[コール]

```
move.l   #srcLen, -(sp)
pea      srcPtr
pea      destPtr
SXCALL   $A1D3
lea      12(sp), sp
```

\$A1D4 GMAdjustPt

ptで指定したポイントがrectPtrで指定したレクタングル内に収まるように調整する。最初からレクタングルの内部にある場合は何もしない。

[引数]

long pt ポイント

long rectPtr レクタングルレコードのアドレス

[返り値]

DO.L 調整した結果のポイント

[コール]

```
pea      rectPtr
```

```

move.l    #pt,-(sp)
SXCALLL   $A1D4
addq.l    #8,sp

```

\$A1D5 GMPutimg

imgPtr で指定したイメージを、カレントグラフポートの rectPtr で指定された位置と大きさで描画する。イメージのタイプはテキストタイプ、描画モードは PSET に固定。描画ページはカレントビットマップのアクセスページで決定される。

[引数]

```

long    imgPtr    イメージのアドレス
long    rectPtr    レクタングルレコードのアドレス

```

[返り値]

```
DO.L    リザルトコード
```

[コール]

```

pea      rectPtr
pea      imgPtr
SXCALLL  $A1D5
addq.l    #8,sp

```

\$A1D6 GMCenterRect

srcRectPtr で指定するレクタングルの中央に、pt で指定する大きさのレクタングルを作成し、destRectPtr に格納する。srcRectPtr がヌルレクタングルの場合、結果もヌルレクタングルとなる。

[引数]

```

long    destRectPtr 結果が返るレクタングルレコードのアドレス
long    srcRectPtr 元になるレクタングルレコードのアドレス
long    pt          新しくつくるレクタングルの大きさ
word    mode        レクタングルがはみ出した場合の処理
                     =0 元になるレクタングルと新しくつくるレクタングルの中心をあわせる
                     =1 新しくつくるレクタングルの左上の座標が必ず元のレクタングルの内側に収まるように

```

する

[返り値]

```

DO.L    =0          結果がヌルレクタングルになった
        =1          正常終了
        =-1         エラー

```

```
AO.L    結果が返るレクタングルレコードのアドレス
```

[コール]

```

move.w    #mode,-(sp)
move.l    #pt,-(sp)
pea        srcRectPtr
pea        destRectPtr
SXCALLL    $A1D6
lea        14(sp),sp

```

\$A1D7 GMScrewRect

rectPtr で指定したレクタングルを外形とする擬似ダイアログをカレントグラフポートに描画する。

レクタングルとして (xs, ys) - (xe, ye) を指定した場合、ビスの位置は (xs+4, ys+4) + (xs+14, ys+14), (xe-15, ys+4) - (xe-5, ys+14), (xs+4, ye-15)-(xs+14, ye-5), (xe-15, ye-15) - (xe-5, ye-5) の4カ所となる。

[引数]

```
long    rectPtr    レクタングルレコードのアドレス
```

[返り値]

```
DO.L    リザルトコード
```

[コール]

```

pea      rectPtr
SXCALLL  $A1D7
addq.l    #4,sp

```

\$A1D8 GMAndRectRgn

srcRectPtr で指定したレクタングルと srcRgnHdl で指定したリージョンの AND をとり、結果を destRgnHdl にリージョンとして格納する。共通部分がない場合、結果はヌルリージョンとなる。

再配置が発生する。

[引数]

```

long    destRgnHdl 結果が返るリージョンレコードへのハンドル
long    srcRgnHdl  リージョンレコードへのハンドル

```

long srcRectPtr レクタングルレコードのアドレス

[返り値]

DO.L リザルトコード

AO.L 結果が返るリージョンレコードへのハンドル

[コール]

pea srcRectPtr

pea srcRgnHdl

pea destRgnHdl

SXCALL \$A1D8

lea l2(sp),sp

\$A1D9 GMPRectRgn

srcRectPtrで指定したレクタングルと srcRgnHdlで指定したリージョンのORをとり、結果を destRgnHdl にリージョンとして格納する。

再配置が発生する。

[引数]

long destRgnHdl 結果が返るリージョンレコードへのハンドル

long srcRgnHdl リージョンレコードへのハンドル

long srcRectPtr レクタングルレコードのアドレス

[返り値]

DO.L リザルトコード

AO.L 結果が返るリージョンレコードへのハンドル

[コール]

pea srcRectPtr

pea srcRgnHdl

pea destRgnHdl

SXCALL \$A1D9

lea l2(sp),sp

\$A1DA GMDiffRectRgn

srcRgnHdlで指定したリージョンの内側で、srcRectPtrで指定したレクタングルの外側の部分を求め、その結果を destRgnHdl にリージョンとして格納する。

再配置が発生する。

[引数]

long destRgnHdl 結果が返るリージョンレコードへのハンドル

long srcRgnHdl リージョンレコードへのハン

ドル

long srcRectPtr レクタングルレコードのアドレス

[返り値]

DO.L リザルトコード

AO.L 結果が返るリージョンレコードへのハンドル

[コール]

pea srcRectPtr

pea srcRgnHdl

pea destRgnHdl

SXCALL \$A1DA

lea l2(sp),sp

\$A1DB GMXorRectRgn

srcRectPtrで指定したレクタングルと srcRgnHdlで指定したリージョンのXORをとり、結果を destRgnHdl にリージョンとして格納する。

再配置が発生する。

[引数]

long destRgnHdl 結果が返るリージョンレコードへのハンドル

long srcRgnHdl リージョンレコードへのハンドル

long srcRectPtr レクタングルレコードのアドレス

[返り値]

DO.L リザルトコード

AO.L 結果が返るリージョンレコードへのハンドル

[コール]

pea srcRectPtr

pea srcRgnHdl

pea destRgnHdl

SXCALL \$A1DB

lea l2(sp),sp

\$A1DC GMCharKind

chで指定した文字の種類を返す。存在しない文字コードを指定した場合、ミッシングキャラクタとなり、全角特殊として分類される。

文字コードの分類は以下のとおり。

文字種コード	文字の種類	コードの範囲
0	半角 ASCII	\$00~FF, \$8000~80FF
1	半角外字	\$F400~F5FF

2	1/4 角上付	\$F000~F1FF
3	1/4 角下付	\$F200~F3FF
4		
5	全角非漢字	\$8140~84BE
6	全角第1水準漢字	\$889F~989E
7	全角第2水準漢字	\$989F~EB9E
8	全角外字	\$EB9F~EC9E
9	全角特殊	\$84BF~889E
		\$F600~FFFF

[引数]

word ch 文字コード(ASCIIコード/
シフト JIS コード)

[返り値]

DO.L 文字種コード

[コール]

```
move.w    # ch, -(sp)
SXCALL    $A1DC
addq.l    # 2, sp
```

\$A1DD GMDiffRgnRect

srcRectPtr で指定したレクタングルの内側で、srcRgnHdl で指定したリージョンの外側の部分を求め、その結果を destRgnHdl にリージョンとして格納する。

再配置が発生する。

[引数]

long destRgnHdl 結果が返るリージョンレコードへのハンドル
long srcRectPtr レクタングルレコードのアドレス
long srcRgnHdl リージョンレコードへのハンドル

[返り値]

DO.L リザルトコード
AO.L 結果が返るリージョンレコードへのハンドル

[コール]

```
pea    srcRgnHdl
pea    srcRectPtr
pea    destRgnHdl
SXCALL $A1DD
lea    12(sp), sp
```

\$A1E0 GMAddFont

fLink で指定したフォントリンクをシステムに追

加する。

[引数]

long fLink フォントリンクのアドレス

[返り値]

DO.L リザルトコード

[コール]

```
pea    fLink
SXCALL $A1E0
addq.l # 4, sp
```

\$A1E1 GMRemoveFont

fKind で指定したフォントカインドとして登録されているフォントを取り外す。

[引数]

long fKind フォントカインド

[返り値]

DO.L リザルトコード

[コール]

```
move.w    # fKind, -(sp)
SXCALL    $A1E1
addq.l    # 2, sp
```

\$A1E2 GMGetFontLink

フォントリンクの先頭が格納されているアドレスを返す。

[引数]

なし

[返り値]

DO.L フォントリンクの先頭が収められているアドレス
AO.L フォントリンクの先頭が収められているアドレス

\$A1E3 GMGetHProcTbl

水平描画の初期化ルーチンのテーブルを返す。

[引数]

なし

[返り値]

DO.L 水平描画の初期化ルーチンのテーブルのアドレス
AO.L 水平描画の初期化ルーチンのテーブルのアドレス

\$A1E6 GMGetStdProcTbl

標準描画ルーチンのテーブルを返す。

[引数]

なし

[返り値]

D0.L 標準描画ルーチンのテーブルのアドレス

A0.L 標準描画ルーチンのテーブルのアドレス

\$A1E7 GMGetFontProcTbl

文字描画ルーチンのテーブルを返す。

[引数]

なし

[返り値]

D0.L 文字描画ルーチンのテーブルのアドレス

A0.L 文字描画ルーチンのテーブルのアドレス

\$A1E8 GMGetRgnProcTbl

リージョン 1 行演算ルーチンのテーブルを返す。

[引数]

なし

[返り値]

D0.L リージョン 1 行演算ルーチンのテーブル
のアドレス

A0.L リージョン 1 行演算ルーチンのテーブル
のアドレス

A

PPENDIX

付録ディスク「SXer Tool Box」の使い方
SX1.10/EasyPaint で追加されたリソース
リザルトコード一覧
SX コール通巻索引

付録ディスク「SXer Tool Box」の使い方

付録ディスクは 2HD, 1.2M バイトの Human 標準形式でフォーマットされていますので, Human, SX-WINDOW とともに問題なく使用できます。収められているファイルのカテゴリーにより, 階層ディレクトリで分類して収録してあります。

¥	
— C 開発キット	C 言語による SX アプリケーション開発キット
— INC	ヘッダファイル等
— LIB	ライブラリ等
— DOC	関数のリファレンス等
— Tools	基本的な開発ツール (アセンブラ, リンカ, リソースリンカ (いずれもフリーソフトウェア))
— SAMPLE_1	「SX-WINDOW プログラミング」掲載のサンプルプログラム
— SAMPLE_2	「追補版 SX-WINDOW プログラミング」掲載のサンプルプログラム
— Freesoftware	フリーソフトウェア
— LHA.X	高圧縮書庫管理プログラム (フリーソフトウェア展開用)
— README	このディスクについてのドキュメントファイル

1 C 開発キットについて

付録ディスクに収録した C 言語による開発キットは, ヘッダファイル, ライブラリ, ドキュメントで構成されるもので, XC2 をお使いの方にはすぐにご利用いただけます。

¥C 開発キット以下のファイルは圧縮されていませんので, そのままご利用いただけます。
インストールの手順は次のとおりです。

- (1) ディレクトリ ¥C 開発キット ¥INC の中のファイルすべてを, 環境変数 include で示されるインクルードサブディレクトリにコピーする。
- (2) ディレクトリ ¥C 開発キット ¥LIB の中のファイルすべてを, 環境変数 lib で示されるライブラリサブディレクトリにコピーする。

¥C 開発キット ¥DOC 中のファイル（筆者が作成したもの）は、用意されている関数のリファレンスです。プリンタで印刷するなどしてご活用ください。

なお、この開発キットは、シャープ株式会社のご好意により収録させていただくものです。

2 サンプルプログラムについて

ディレクトリ ¥SAMPLE_1, ¥SAMPLE_2 以下のファイルは、それぞれ『SX-WINDOW プログラミング』、『追補版 SX-WINDOW プログラミング』に収録したサンプルプログラムのソースを収録したものです。

¥SAMPLE_1, ¥SAMPLE_2 以下のファイルは圧縮されていませんので、そのままご利用いただけます。

3 基本的な開発ツールについて

ディレクトリ ¥Tools 以下のファイルは、SX アプリケーションの開発に最低限必要なアセンブラ、リンカ、そしてリソースリンカのセットです。

・アセンブラ

「ハイスピードアセンブラ・HAS」は YuNK 氏（ハンドル名）の手によるもので、純正の AS.X バージョン 2 の上位互換です。純正アセンブラよりも高速で、いくつかの便利な機能が付加されています。

・リンカ

「ハイスピードリンカ・HLK」は SALT 氏（ハンドル名）製作のもので、純正の LK.X バージョン 2 の上位互換です。純正リンカと比較すると非常に高速です。

・リソースリンカ

「RLK」は清水和久氏製作のもので、リソースの追加、抽出、参照等が可能です。

以上のソフトウェアはいずれもフリーソフトウェアで、LHA で圧縮してあります。

LHA は高圧縮書庫管理プログラム、いわゆるアーカイバで、複数のファイルをまとめて圧縮（サイズを小さくすること）するためのツールです。LHA によって圧縮された複数のファイルは、拡張子に .LZH という名前のついた 1 つのファイルにまとめられます。また、こうしたファイルは LHA を使って元の複数のファイルに復元することができます。これを展開、あるいは解凍と呼びます。

ハイスピードアセンブラの場合、HAS234.LZH というファイル名で収録されています。これを展開する場合の手順を示します。

- (1) あらかじめフォーマットしておいた空ディスクを 1 枚用意しておきます。これをドライブ B に入れます。このディスクにファイルが展開されることになります。ハードディ

スクや RAM ディスクでもかまいませんが、この場合、以降に示すドライブ名を、お手持ちのマシンのドライブ名に読み替えてください。

- (2) 付録ディスクをドライブ A に入れます。
- (3) B: [F1] 等で展開するドライブに移動する。
- (4) A: ¥LHA x A: ¥Tools¥HAS234 [F1] と入力する。これで LHA が働きはじめ、ドライブ B の中にハイスピードアセンブラと、それに付随するドキュメントファイル等が作成されます。

4 フリーソフトウェアについて

ディレクトリ ¥Freesoftware には 13 作品、14 ファイルのフリーソフトウェアが収められています。個々のフリーソフトウェアについての説明は、ディスク中の README ドキュメントを参照してください。

これらのフリーソフトウェアはいずれも LHA で圧縮してあります。ご利用に際しては、前節で示した例のようにして、展開してご利用ください。

付録ディスク「SXer Tool Box」には、以下のフリーソフトウェアを収録させていただきました。

HAS234.LZH	YuNK 氏（ハンドル名）製作
純正アセンブラ AS.X と上位互換のアセンブラです。アセンブル速度が向上しているほか、いくつかの便利な機能が拡張されています	
hlk212.lzh	SALT 氏（ハンドル名）製作
純正リンカ LK.X と上位互換のリンカです。リンク速度が大幅に向上しています	
rsc102.lzh	清水和久氏製作
『oh!X』誌 '90 年 1 月号の付録ディスクに収録されていた、純正リソースリンカ RLK と上位互換のリソースリンカです。リソースの取り出し等の機能が拡張されています	
clip100.lzh	沖@沖氏（ハンドル名）製作
グラフィックのカット & ペーストに対応したクリップボードです	
HyperMenu.lzh	DAI 氏（ハンドル名）製作
SX-WINDOW 上の機能を登録し、使いやすく提供する階層メニューを実現するユーティリティです	
neko023x.lzh neko023s.lzh	ひとり氏（ハンドル名）製作
ウィンドウ内を「猫」がマウスカーソルを追って走り回るデスクアクセサリです neko023x.lzh には実行ファイル、neko023s.lzh にはソースが収められています	
SXBGM140.LZH	GRANADA 氏（ハンドル名）製作
「サウンド.X」上位互換の OPM/PCM ファイル演奏アクセサリです。連続演奏等の機能が拡張されています	

SXcon005.lzh	SALT 氏（ハンドル名）製作
SX-WINDOW 上で Human の標準入出力を利用できるコンソールのウィンドウを実現する console.x, そこで command.x を利用するための shell.x が収められています	
SXEVENTS.LZH	Arimac 氏（ハンドル名）製作
SX-WINDOW 上で発生するイベントを監視し、その内容を表示するためのユーティリティーです	
SXmxp301.Lzh sxmxlvb2.Lzh	さとし氏（ハンドル名）製作
SXmxp.X はフリーソフトウェアとして広く利用されている音楽演奏システム MXDRV のデータを SX-WINDOW 上で演奏させるためのアクセサリ。sxmxlvb.X は、その演奏状態を表示するレベルメータのアクセサリです	
SXtask.Lzh	OUZAK 氏（ハンドル名）製作
シェル上で動作するタスクにブレークポイントを仕掛けるツールです。デバッグと併用することにより、シェル上で動作するアプリケーションのデバッグに威力を発揮します	
OpIt091.lzh	勝呂友一氏作
実行ファイルを登録することで、キーまたはメニュー発でプログラムの起動を可能にするユーティリティーです	
LHA_X624.x	岡田紀雄氏作
ルートに収められた LHA.x のオリジナルアーカイブファイルです。自己解凍ファイルとなっており、このファイルを実行すると、ドキュメントなどが展開されます	
mkBG120.Lzh	拙作
512×512 ドット 65536 色のグラフィックを 4 階調変換し、SX-WINDOW の背景となるデータを作成するユーティリティーです	
St0105a.Lzh	拙作
実画面モードで SX-WINDOW のデスクトップをマウスポインタに追従してスクロールさせるユーティリティーです	

5 ディスクの著作権等について

日本の著作権法には「著作権の放棄」という概念がないため、このディスクに収められている各ファイルの著作権はそれぞれのファイルの作者等に帰属します。ただし、各ファイルの著作権保持者はいずれも再配布を認めていますので、このディスクは自由に複写して再配布することができます。ただし、シャープ株式会社から提供された C 開発キットの著作権は同社が有し、フリーソフトウェアではないことをお断わりしておきます。

サンプルプログラムについては、ご自由に改変し、SX-WINDOW への理解を深めていただければ幸いです。また、改変して作成したプログラムの著作権は、改変した方に帰属します。

フリーソフトウェアについては、各フリーソフトウェアごとに作者からのドキュメントファイルが付属しているはずですので、再配布や改変等に際しては、それらのファイルをよく読み、従うようにしてください。

6

そのほか

そのほかにも重要な情報がディスク中の README ファイル中に収められていますので、ご使用前には必ず README ファイルもあわせてお読みくださいますよう、お願いします。

SX1.10/EasyPaint で追加されたリソース

行末のマークは、このリソースが収められているリソースファイルを示す。

S	SYSTEM.LB
B	BUILTIN.LB
I	ICON.LB
C	コントロール.LB (旧 CTRLPNL.LB)
E	Easypaint.LB

Easypaint で使用されているリソースについては、リソースの意味と使用されている ID だけを示す。

WDEF	ウィンドウ定義関数を収めるリソース 49 タイトルの広い標準ウィンドウ (クローズボタン, スクロールバー, サイズボタンをサポート) S
MENU	メニューテンプレートを収めるリソース -4096 テキストエディット用ポップアップメニュー S
PRTD	プリンタドライバを収めるリソース 0 CZ 系 24 ビン S 1 ESC/P 系 S 2 PC-PR 系 S 3 CZ 系 48 ビン S
PRIC	プリンタのレクタングルイメージ (2 ページ) を収めるリソース (未使用) 0 CZ 系 24 ビン S 1 ESC/P 系 S 2 PC-PR 系 S 3 CZ 系 48 ビン S
PrEV	デフォルトの印刷環境レコードの内容を収めるリソース 0 デフォルトの印刷環境レコードの内容 B
ICN #	アイコン定義レコードを収めるリソース 165 Easypaint.X I 166 *.PNT I 167 Easyscan.X I 168 Easyprint.X I
PAT4	3 ページのレクタングルイメージ (マスク付き) を収めるリソース 222 E
PAT3	2 ページのレクタングルイメージ (マスク付き) を収めるリソース 165 Easypaint.X I 166 *.PNT I 167 Easyscan.X I 168 Easyprint.X I 1000 E

APPENDIX

PAT2	2 ページのレクタングルイメージを収めるリソース	
	-3155 画面設定	C
	-3156 キーボード設定	C
	-3157 マウス設定	C
	-3158 プリンタ設定	C
	-3159 スイッチ設定	C
	-3160 RS-232C 設定	C
	-3161 タイマー設定	C
	-3162 印刷環境設定	C
	-217 ボタン	C
	-218 押されたボタン	C
	1000~1005, 1010~1022, 1040~1057, 1060~1067, 1100	E
	217~221, 223~242, 250~253	E
PAT1	1 ページのレクタングルイメージを収めるリソース 1000, 1001	E
CODE	プログラムを収めるリソース -3162 プリンタ設定	C
CPNM	タスク名 (ASCII) を収めるリソース -3155 画面設定 -3156 キーボード設定 -3157 マウス設定 -3158 プリンタ設定 -3159 スイッチ設定 -3160 RS-232C 設定 -3161 タイマー設定 -3162 印刷環境設定	C C C C C C C C C
MCSR	マウスポインタの形状を収めるリソース 1000~1011	E
PaPT	ペンパターンを収めるリソース 1000	E
PaBR	ブラシパターンを収めるリソース 1000	E
PaSP	スプレーパターンを収めるリソース 1000	E
PaMS	1000	E
PaDD	Easypaint の変更可能なデータ (文字列等) が収められるリソース 1000~1005	E

リザルトコード一覧

””で囲んだ文章は、リザルトコードに対応するシェル用エラーダイアログの表示です。

一般		
	\$00000000	正常終了
	\$FFFFFFFF	エラー
メモリマネージャ		
ER_NOMEM	\$FFFFFFC00	ヒープの拡張に失敗した
ER_OFFADR	\$FFFFFFC01	奇数アドレスを指定した
ER_ZONEID	\$FFFFFFC02	ゾーンヘッダが異常
ER_NILPTR	\$FFFFFFC03	ポインタが空 (NULL)
ER_NILHDL	\$FFFFFFC04	ハンドルが空 (NULL)
ER_EMPHDL	\$FFFFFFC05	ハンドルの指しているブロックのサイズが 0
ER_NOTFRE	\$FFFFFFC06	フリーブロックでない
ER_NOTALO	\$FFFFFFC07	アロケートッドブロックでない
ER_NOTNON	\$FFFFFFC08	再配置不能ブロックではない
ER_NOTREL	\$FFFFFFC09	再配置可能ブロックではない
ER_NOTLOC	\$FFFFFFC0A	ロックされたブロックではない
ER_NOTUNL	\$FFFFFFC0B	ロックされていないブロックではない
ER_NOTPUR	\$FFFFFFC0C	ページ可能ブロックではない
ER_NOTUNP	\$FFFFFFC0D	ページ不可能なブロックではない
ER_ILPROP	\$FFFFFFC0E	属性の指定が異常
ER_DIFTYP	\$FFFFFFC0F	フリーブロックとアロケートッドブロックをまとめようとした (連続する 2 つのブロックの種類が異なる)
ER_LESSIZ	\$FFFFFFC10	ヒープゾーンの範囲が異常
ER_SPLIT	\$FFFFFFC11	フリーブロックのサイズが足りない (ブロック分割の失敗)
ER_SIZEPU	\$FFFFFFC12	ページ可能ブロックのサイズを変更しようとした
リソースマネージャ		
ER_RSCNOTFND	\$FFFFFF800	リソースを発見できなかった
ER_EXISTYPE	\$FFFFFF7FF	すでにそのタイプは存在する
ER_EXISTID	\$FFFFFF7FE	すでにそのタイプの ID は存在する
ER_TYPENOTFND	\$FFFFFF7FD	このタイプは存在しない
ER_IDNOTFND	\$FFFFFF7FC	この ID は存在しない
ER_ILLTYPE	\$FFFFFF7FB	タイプに” ”を指定した
ER_ILLID	\$FFFFFF7FA	ID に \$FFFF を指定した
ER_NILCURRENT	\$FFFFFF7F9	カレントリソースが設定されていない
ER_NOTOPEN	\$FFFFFF7F8	リソースはすべてメモリ上にある (ファイルがオープンされていない)
ER_NILHANDLE	\$FFFFFF7F7	ハンドルが空
ER_HDLNOTFND	\$FFFFFF7F6	このハンドルに該当するリソースはない
ER_CANTDETATCH	\$FFFFFF7F5	ハンドルが正しくない

テキストマネージャ		
TM_EDITABORT	\$FFFFD800	致命的なエラーが発生し編集テキストを廃棄した
TM_LINEOVER	\$FFFFD801	指定されたレクタングルに文字が収まらない
TM_LENOWER	\$FFFFD802	テキストの最大サイズを超えた
TM_PROHIBITEDIT	\$FFFFD803	リードオンリーで編集できない
TM_EDITERR	\$FFFFD804	不正な文字列を入力した
タスクマネージャ		
ER_ABORT	\$FFFFDFFE	ハードウェアエラーによるアボート
ER_OBJX	\$FFFFDFFF	単独実行のファイル
ER_NOTHEAD	\$FFFFE000	タスクマネージャ上で動作しないファイル
ER_NOTOBJECT	\$FFFFE001	実行ファイルではない
ER_NOTLOAD	\$FFFFE002	ファイルオープンエラー
ER_NOTPARA	\$FFFFE003	同時に複数実行できないファイル
	\$FFFFE004	"ディスクの書き込みが禁止されています。コピーできません。"
	\$FFFFE005	"読み込み専用のファイルです。"
	\$FFFFE006	"同名の読み込み専用ファイルがあります。コピーできません。"
	\$FFFFE007	"読み込み専用のファイルです。変更できません。"
	\$FFFFE008	"システム属性のファイルです。移動/削除はできません。"
	\$FFFFE009	"同名のシステム属性ファイルがあります。コピーできません。"
	\$FFFFE00A	"システム属性のファイルです。変更できません。"
	\$FFFFE00B	"同名のファイルがあるので削除します。よろしいですか?"
	\$FFFFE00C	"ドライブ<DRVNAME>の容量が<LENGTH>Kバイト不足しています。"
	\$FFFFE00D	ファイル名が正しくない
	\$FFFFE00E	"同名のファイルが存在します。"
	\$FFFFE00F	"アイコンデータに変更があります。ファイルに保存しますか?"
	\$FFFFE010	"終了します。"
	\$FFFFE011	元の場所ではない
	\$FFFFE012	"メモリの空き容量がありません。ウィンドウをクローズしてください。"
	\$FFFFE013	"メモリの空き容量がありません。ウィンドウを1つクローズします!!"
	\$FFFFE014	"ディスクの書き込みが禁止されています。"
	\$FFFFE015	"複数のファイルは同時に実行できません。"
	\$FFFFE016	"メモリ容量が足りません終了します。"
	\$FFFFE017	"ディレクトリが作成できません。"
	\$FFFFE018	"ファイルの書き込みに失敗しました。"
	\$FFFFE019	"オープン中のファイルがあります。"
	\$FFFFE01A	"ウィンドウをクローズしなければ、強制終了します"
	\$FFFFE01B	"メモリ容量が足りません再起動します。"
	\$FFFFE01C	"メモリの空き容量に余裕ができました。"
	\$FFFFE01D	"パス名が長すぎます。(64バイトまで)"
	\$FFFFE01E	"リソースの読み込みに失敗しました。"
ER_FILENOTFND	\$FFFFE01F	ファイルが見つからない
ER_SERCHBREAK	\$FFFFE020	ファイル検索が中断された
	\$FFFFE021	指定されたファイルが見つからない/1 ドライブ検索した
	\$FFFFE022	ドライブの準備ができていない

SX コール通巻索引

前著『SX-WINDOW～』の第4章および本書の第5章に掲載されているSXコールの通巻索引です。索引の中で使われている記号の意味は次のとおりです。

- I ……『SX-WINDOW～』の掲載ページ
 II ……本書の掲載ページ
 ◎ ……SX1.10 になって仕様が変更または新設されたコール
 (ページ数) ……SX1.02 では未公開であったが SX1.10 になり正式に公開された
 コールが『SX-WINDOW～』で解説されているページ

コール番号順索引

\$A000	MMInitHeap	I 315	\$A023	MMChPurge	I 320
\$A001	MMGetCurrentHeap	I 315	\$A024	MMChMelt	I 320
\$A002	MMSetCurrentHeap	I 315	\$A025	MMChReserve	I 321
\$A003	MMNewHandle	I 315	\$A026	MMChFreeSize	I 321
\$A004	MMSetHandleSize	I 315	\$A027	MMChGrowHeapGet	I 321
\$A005	MMDisposeHandle	I 315	\$A028	MMChGrowHeapSet	I 321
\$A006	MMGetHandleSize	I 316	\$A029	MMChPurgeGet	I 321
\$A007	MMHLock	I 316	\$A02A	MMChPurgeSet	I 321
\$A008	MMHUnlock	I 316	\$A02B	MMChCompactGet	I 321
\$A009	MMNewPtr	I 316	\$A02C	MMChCompactSet	I 321
\$A00A	MMDisposePtr	I 316	\$A02D	MMPtrNew	I 321
\$A00B	MMGetPtrSize	I 316	\$A02E	MMPtrHeap	I 322
\$A00C	MMSetPtrSize	I 316	\$A02F	MMPtrDispose	I 322
\$A00D	MMCompactMem	I 317	\$A030	MMPtrSizeGet	I 322
\$A00E	MMHeapInit	I 317	\$A031	MMPtrSizeSet	I 322
\$A00F	MMBlockMstGet	I 317	\$A032	MMPtrPropGet	I 322
\$A010	MMMemCompact	I 317	\$A033	MMPtrPropSet	I 322
\$A011	MMMemPurge	I 317	\$A034	MMMstAllocate	I 323
\$A012	MMMemMelt	I 318	\$A035	MMMstBind	I 323
\$A013	MMMemReserve	I 318	\$A036	MMHdlNew	I 323
\$A014	MMMemSizeFree	I 318	\$A037	MMHdlHeap	I 323
\$A015	MMMemSizeComp	I 318	\$A038	MMHdlDispose	I 323
\$A016	MMMemSizePurge	I 318	\$A039	MMHdlSizeGet	I 323
\$A017	MMMemSizeMelt	I 318	\$A03A	MMHdlSizeSet	I 324
\$A018	MMMemErrorGet	I 319	\$A03B	MMHdlEmpty	I 324
\$A019	MMMemErrorSet	I 319	\$A03C	MMHdlRealloc	I 324
\$A01A	MMMemStrictGet	I 319	\$A03D	MMHdlMoveHi	I 324
\$A01B	MMMemStrictSet	I 319	\$A03E	MMHdlPropGet	I 324
\$A01C	MMChGet	I 319	\$A03F	MMHdlPropSet	I 324
\$A01D	MMChSet	I 319	\$A040	MMHdlLock	I 325
\$A01E	MMChPtrNew	I 319	\$A041	MMHdlUnlock	I 325
\$A01F	MMChMstMore	I 320	\$A042	MMHdlPurge	I 325
\$A020	MMChMstNew	I 320	\$A043	MMHdlNoPurge	I 325
\$A021	MMChHdlNew	I 320	\$A044	MMHdlResource	I 325
\$A022	MMChCompact	I 320	\$A045	MMHdlNoResource	I 325

APPENDIX

\$A046	MMHdlIns	I 326	\$A0A2	EMInit	I 334
\$A047	MMHdlDel	I 326	\$A0A3	EMTini	I 334
\$A048	MMBlockUsrFlagGet	I 326	\$A0A4	EMSet	I 334
\$A049	MMBlockUsrFlagSet	I 326	\$A0A5	EMGet	I 335
\$A04A	MMBlockUsrWordGet	I 326	\$A0A6	EMScan	I 335
\$A04B	MMBlockUsrWordSet	I 326	\$A0A7	EMMSLoc	I 335
\$A04C	MMMemAmiTpeach	I 327	\$A0A8	EMLBttn	I 335
\$A04D	MMMemHiReserve	I 327	\$A0A9	EMRBttn	I 335
\$A04E	MMPtrBlock	I 327	\$A0AA	EMLStill	I 335
\$A04F	MMHdlBlock	I 327	\$A0AB	EMRStill	I 335
\$A050	MMHdlMstGet	I 327	\$A0AC	EMLWait	I 335
\$A051	MMChHiReserve	I 328	\$A0AD	EMRWait	I 336
\$A052	MMChUsrFlagGet	I 328	\$A0AE	EMKMapGet	I 336
\$A053	MMChUsrFlagSet	I 328	\$A0AF	EMSystime	I 336
\$A054	MMChUsrWordGet	I 328	\$A0B0	EMDClickGet	I 336
\$A055	MMChUsrWordSet	I 328	\$A0B1	EMBlinkGet	I 336
\$A068	EXEnVDISPST	I 328	\$A0B2	EMClean	I 336
\$A069	EXDeVDISPST	I 328	\$A0B3	EMMaskSet	I 336
\$A06A	MSInitCsr	I 329	\$A0B4	EMDTSkSet	I 336
\$A06B	MSShowCsr	I 329	\$A0B5	EMDClickSet	I 337
\$A06C	MSHideCsr	I 329	\$A0B6	EMBlinkSet	I 337
\$A06D	MSSetCsr	I 329	\$A0B7	EMEnCross	I 337
\$A06E	MSObscureCsr	I 329	\$A0B8	EMDeCross	I 337
\$A06F	MSShieldCsr	I 329	\$A0D9	RMInit	I 337
\$A070	MSGetCurMsr	I 330	\$A0DA	RMTini	I 337
\$A071	MSMultiGet	I 330	\$A0DB	RMResNew	I 337
\$A072	MSMultiSet	I 330	\$A0DC	RMRscAdd	I 337
\$A073	EXAnimStart	I 330	\$A0DD	RMRscRemove	I 338
\$A074	EXAnimEnd	I 330	\$A0DE	RMTypRemove	I 338
\$A075	EXAnimTest	I 330	\$A0DF	RMResDispose	I 338
\$A086	KBMapGet	I 331	\$A0E0	RMResOpen	I 338
\$A087	KBShiftGet	I 331	\$A0E1	RMRscGet	I 338
\$A088	KBShiftSet	I 331	\$A0E2	RMResClose	I 338
\$A089	KBSimulate	I 332	\$A0E3	RMResRemove	I 339
\$A08A	KBScan	I 332	\$A0E4	RMCurResSet	I 339
\$A08B	KBGet	I 332	\$A0E5	RMRscRelease	I 339
\$A08C	KBEmpty	I 331	\$A0E6	RMRscDetach	I 339
\$A08D	KBInit	I 331	\$A0E7	RMMaxIDGet	I 339
\$A08E	KBTini	I 331	\$A0E8	RMResSave	I 339
\$A08F	KBCurKbrGet	I 332	\$A0E9	RMHdlToRsc	I 339
\$A090	KBOldOnGet	I 332	\$A0EA	RMCurResGet	I 340
\$A091	KBOldOnSet	I 332	\$A0EB	RMLastResGet	I 340
\$A092◎	KBFlagGet	II 275	\$A0EC	RMResLoad	I 340
\$A093◎	KBFlagSet	II 275	\$A0ED◎	RMResLinkGet	II 275
\$A09A	KMEmpty	I 333	\$A0EE◎	RMResTypeList	II 275
\$A09B	KMPost	I 333	\$A0EF◎	RMResIDList	II 276
\$A09C	KMAscJobSet	I 333	\$A12D	GMOpenGraph	I 340
\$A09D	KMSimulate	I 333	\$A12E	GMCloseGraph	I 340
\$A09E	KMTask	I 333	\$A130	GMInitGraph	I 341
\$A09F	KMInit	I 333	\$A131	GMSetGraph	I 341
\$A0A0	KMTini	I 334	\$A132	GMGetGraph	I 341
\$A0A1	KMCurKmrGet	I 334	\$A133	GMCopyGraph	I 341

\$A135	●	I 341	\$A169	GMRectInRgn	I 351
\$A136	GMMoveGraph	I 341	\$A16A	GMEqualRgn	I 351
\$A137	GMSlideGraph	I 342	\$A16B	GMEEmptyRgn	I 351
\$A138	GMSetClip	I 342	\$A16C◎	GMImgToRgn	II 305
\$A139	GMGetClip	I 342	\$A16D	GMInitBitmap	I 352
\$A13A	GMClipRect	I 342	\$A16E	GMMove	I 352
\$A13B	GMSetHome	I 342	\$A16F	GMMoveRel	I 352
\$A13C	GMSetGraphSize	I 342	\$A170	GMLine	I 352
\$A13D	GMSetBitmap	I 343	\$A171	GMLineRel	I 352
\$A13E	GMLocalToGlobal	I 343	\$A172	GMFrameRect	I 352
\$A13F	GMGlobalToLocal	I 343	\$A173	GMFillRect	I 353
\$A140	GMInitPen	I 343	\$A174	GMFrameOval	I 353
\$A141	GMPenShow	I 343	\$A175	GMFillOval	I 353
\$A142	GMPenHide	I 343	\$A176	GMFrameRRect	I 353
\$A143	GMPenSize	I 343	\$A177	GMFillRRect	I 353
\$A144	GMPenMode	I 344	\$A178◎	GMFrameArc	II 305
\$A145	GMPenPat	I 344	\$A179◎	GMFillArc	II 305
\$A146	GMExPat	I 344	\$A17A	GMFrameRgn	I 353
\$A147	GMForeColor	I 344	\$A17B	GMFillRgn	I 354
\$A148	GMBBackColor	I 344	\$A17C◎	GMFramePoly	II 306
\$A149	GMAPage	I 345	\$A17D◎	GMFillPoly	II 306
\$A14A	GMGetLoc	I 345	\$A17E	GMScroll	I 354
\$A14B	GMGetPen	I 345	\$A17F	GMCopy	I 354
\$A14C	GMSetPen	I 345	\$A180	GMCopyMask	I 354
\$A14D	GMInitialize	I 345	\$A181	●	I 355
\$A14E	GMNullRect	I 345	\$A182	GMPlotImg	I 355
\$A14F	GMSizeRect	I 345	\$A183	GMPutRImg	I 355
\$A150	GMAAndRects	I 346	\$A186	GMDupHImg	I 356
\$A151	GMMoveRect	I 346	\$A187	GMDupVImg	I 356
\$A152	GMSlideRect	I 346	\$A188	GMDupHRImg	I 356
\$A153	GMInsetRect	I 346	\$A189	GMDupVRImg	I 356
\$A154	GMAAndRect	I 347	\$A18B	GMFontKind	I 357
\$A155	GMOrRect	I 347	\$A18C	GMFontFace	I 357
\$A156	GMPTInRect	I 347	\$A18D	GMFontMode	I 357
\$A157	GMEqualRect	I 347	\$A18E	GMFontSize	I 357
\$A158	GMEEmptyRect	I 348	\$A18F	GMDrawChar	I 357
\$A159	GMAdjustRect	I 348	\$A190	GMDrawStrL	I 358
\$A15A	GMNewRgn	I 348	\$A191	GMDrawStr	I 358
\$A15B	GMDisposeRgn	I 348	\$A192	GMDrawStrZ	I 358
\$A15C	GMOpenRgn	I 348	\$A194	GMCharWidth	I 358
\$A15D	GMCloseRgn	I 348	\$A195	GMStrLWidth	I 358
\$A15E	GMNullRgn	I 349	\$A196	GMStrWidth	I 359
\$A15F	GMRectRgn	I 349	\$A197	GMStrLength	I 359
\$A160	GMCopyRgn	I 349	\$A198	GMFontInfo	I 359
\$A161	GMMoveRgn	I 349	\$A199	GMOpenScript	I 359
\$A162	GMSlideRgn	I 349	\$A19A	GMCloseScript	I 359
\$A163	GMInsetRgn	I 349	\$A19B	GMDisposeScript	I 360
\$A164	GMAAndRgn	I 350	\$A19C	GMDrawScript	I 360
\$A165	GMOrRgn	I 350	\$A19D	GMGetScript	I 360
\$A166	GMDiffRgn	I 350	\$A19E◎	GMOpenPoly	II 306
\$A167	GMXorRgn	I 350	\$A19F◎	GMClosePoly	II 306
\$A168	GMPTInRgn	I 351	\$A1A0◎	GMDisposePoly	II 306

APPENDIX

\$AIA1	GMShadowStrZ	I 360	\$AID8○	GMAAndRectRgn	II 312
\$AIA2	GMShadowRect	I 361	\$AID9○	GMOOrRectRgn	II 313
\$AIA3	GMInvertRect	I 361	\$AIDA○	GMDiffRectRgn	II 313
\$AIA5	GMInvertBits	I 361	\$AIDB○	GMXorRectRgn	II 313
\$AIA6	GMMMapPt	I 361	\$AIDC○	GMCharKind	II 313
\$AIA7	GMMMapRect	I 361	\$AIDD○	GMDiffRgnRect	II 314
\$AIA8○	GMMMapPoly	II 306	\$AIE0○	GMAddFont	II 314
\$AIA9	GMMMapRgn	I 362	\$AIE1○	GMRemoveFont	II 314
\$AIAA	GMScalePt	I 362	\$AIE2○	GMGetFontLink	II 314
\$AIAB	GMInitPalet	I 362	\$AIE3○	GMGetHProcTbl	II 314
\$AIAC	●	I 363	\$AIE6○	GMGetStdProcTbl	II 314
\$AIAD	GMDrawG16	I 363	\$AIE7○	GMGetFontProcTbl	II 314
\$AIAE	●	I 363	\$AIE8○	GMGetRgnProcTbl	II 314
\$AIAF	GMGetPixel	I 363	\$AIF8	WMInit	I 366
\$AIB1	GMCalcMask	I 363	\$AIF9	WMOpen	I 366
\$AIB2	GMCalcFrame	I 364	\$AIFA	WMRefer	I 366
\$AIB3	SXLongMul	I 364	\$AIFB	WMClose	I 367
\$AIB4	SXFixRound	I 364	\$AIFC	WMDispose	I 367
\$AIB6	SXFixMul	I 364	\$AIFD	WMFind	I 367
\$AIB7	SXFixDiv	I 365	\$AIFE	WMSelect	I 367
\$AIB8	GMGetFontTable	I 365	\$AIFF○	WMSelect2	(I 367), II 276
\$AIB9○	GMCopyStdProc	II 307	\$A200	WMCarry	I 368
\$AIBA	GMStrZWidth	I 365	\$A201	WMShine	I 368
\$AIBB○	GMTransImg	II 307	\$A202	WMMove	I 368
\$AIBC○	GMFillRImg	II 307	\$A203	WMSize	I 368
\$AIBD○	GMFillImg	II 307	\$A204	WMGrow	I 369
\$AIBE○	GMSlidedRgn	II 308	\$A205	WMDrag	I 369
\$AIBF○	GMPaintRgn	II 308	\$A206	WMZoom	I 369
\$AIC0○	GMSetRgnLine	II 308	\$A207	WMShow	I 369
\$AICI○	GMGetRgnLine	II 308	\$A208	WMHide	I 370
\$AIC2○	GMInitGraphMode	II 308	\$A209	WMShowHide	I 370
\$AIC3○	GMCurFont	II 309	\$A20A	WMCheckBox	I 370
\$AIC4○	GMGetScrnSize	II 309	\$A20B	WMCheckCBox	I 370
\$AIC5○	GMExgGraph	II 309	\$A20C	WMDrawGBox	I 370
\$AIC6○	GMExgBitmap	II 309	\$A20D	WMUpdate	I 371
\$AIC7○	GMGetBitmap	II 309	\$A20E	WMUpdtOver	I 371
\$AIC8○	GMCalcBitmap	II 309	\$A20F	WMActive	I 371
\$AIC9○	GMCalcScrnSize	II 309	\$A210	WMGraphGet	I 371
\$AICA○	GMNewBits	II 310	\$A211	●	I 371
\$AICB○	GMDisposeBits	II 310	\$A212	●	I 371
\$AICC○	GMLockBits	II 310	\$A213	●	I 371
\$AICD○	GMUnlockBits	II 310	\$A214	●	I 372
\$AICE○	GMItalicRect	II 310	\$A215	●	I 372
\$AICF○	GMItalicRgn	II 310	\$A216	●	I 372
\$AID0○	GMFreeBits	II 311	\$A217	●	I 372
\$AID1○	GMCalcGraph	II 311	\$A218	WMAddRect	I 372
\$AID2○	GMPackImage	II 311	\$A219	WMAddRgn	I 373
\$AID3○	GMUnpackImage	II 311	\$A21A	WMSubRect	I 373
\$AID4○	GMAjustPt	II 311	\$A21B	WMSubRgn	I 373
\$AID5○	GMPutImg	II 312	\$A21C	WMGScriptSet	I 373
\$AID6○	GMCenterRect	II 312	\$A21D	WMGScriptGet	I 373
\$AID7○	GMscrewRect	II 312	\$A21E	WMTITLESet	I 373

\$A21F	WMTitleGet	I 373	\$A2C3	DMOpen	I 381
\$A220	WMTIDSet	I 374	\$A2C4	DMRefer	I 382
\$A221	WMTIDGet	I 374	\$A2C5	DMClose	I 382
\$A222	WMPinRect	I 374	\$A2C6	DMDDispose	I 382
\$A223	WMCalcUpdt	I 374	\$A2C7	DMControl	I 382
\$A224	WMGetDTGS	I 374	\$A2C8	DMDDraw	I 382
\$A225	WMDragRgn	I 374	\$A2C9	Alert	I 383
\$A227◎	WSOpen	II 279	\$A2CA	StopAlert	I 383
\$A228◎	WSClose	II 279	\$A2CB	NoteAlert	I 383
\$A229◎	WSDispose	II 279	\$A2CC	CautionAlert	I 383
\$A22A◎	WSEnlist	II 279	\$A2CD	CouldAlert	I 383
\$A22B◎	WSDelist	II 279	\$A2CE	FreeAlert	I 384
\$A22C◎	WMOptionGet	II 276	\$A2CF	DIGet	I 384
\$A22D◎	WMOptionSet	II 276	\$A2D0	DISet	I 384
\$A266	MNInit	I 375	\$A2D1	DITGet	I 384
\$A267	MNRefer	I 375	\$A2D2	DITSet	I 385
\$A268	MNSelect	I 375	\$A2D3	DITSelect	I 385
\$A269◎	MNConvert	II 278	\$A2D4	GetAlrtStage	I 385
\$A289	CMOpen	I 376	\$A2D5	ResetAlrtStage	I 385
\$A28A	CMDDispose	I 376	\$A2D6	DIUpdate	I 385
\$A28B	CMKill	I 376	\$A2D7	DMBeep	I 385
\$A28C	CMHide	I 376	\$A2D8	DIHide	I 385
\$A28D	CMShow	I 377	\$A2D9	DIShow	I 386
\$A28E	CMDraw	I 377	\$A2F6	DMEError	I 386
\$A28F	CMDDrawOne	I 377	\$A2F7	DMWaitOpen	I 386
\$A290	CMValueSet	I 377	\$A2F8	DMWaitClose	I 386
\$A291	CMValueGet	I 377	\$A2F9	DMWaitWhile	I 386
\$A292	CMMinSet	I 377	\$A30A	TMInit	I 387
\$A293	CMMinGet	I 378	\$A30B	TMNew	I 387
\$A294	CMMMaxSet	I 378	\$A30C	TMSetRect	I 387
\$A295	CMMMaxGet	I 378	\$A30D	TMChangText	I 388
\$A296	CMMove	I 378	\$A30E	TMIdle	I 388
\$A297	CMSize	I 378	\$A30F	TMActive	I 388
\$A298	CMShine	I 378	\$A310	TMDeactive	I 388
\$A299	CMFind	I 379	\$A311	TMCarat	I 388
\$A29A	CMCheck	I 379	\$A312	TMDispose	I 388
\$A29B	CMRefer	I 379	\$A313	TMUpDate	I 389
\$A29C	CMTTitleGet	I 380	\$A314	TMSetText	I 389
\$A29D	CMDragControl	I 380	\$A315	TMGetText	I 389
\$A29E	CMDraws	I 380	\$A316	TMSetSelect	I 389
\$A29F	CMTTitleSet	I 380	\$A317◎	TMKey	(I 390), II 285
\$A2A0◎	CMOptionGet	II 277	\$A318◎	TMStr	(I 390), II 285
\$A2A1◎	CMOptionSet	II 277	\$A319◎	TMCalText	(I 390), II 285
\$A2A2◎	CMUserGet	II 277	\$A31A	TMPinScroll	I 390
\$A2A3◎	CMUserSet	II 277	\$A31B	TMClick	I 390
\$A2A4◎	CMProcGet	II 277	\$A31C◎	TMEvent	(I 391), II 285
\$A2A5◎	CMProcSet	II 277	\$A31D	●	I 391
\$A2A6◎	CMDefDataSet	II 278	\$A31E	●	I 392
\$A2A7◎	CMDefDataSet	II 278	\$A31F	●	I 392
\$A2C0	DMInit	I 380	\$A320◎	TMCut	(I 392), II 286
\$A2C1	ErrorSound	I 381	\$A321	TMCopy	I 392
\$A2C2	DMFontSet	I 381	\$A322◎	TMPaste	(I 392), II 286

APPENDIX

\$A323◎	TMDelete	(I 393),II 286	\$A35B	TSGetTdb	I 400
\$A324◎	TMInsert	(I 393),II 286	\$A35C	TSSetTdb	I 400
\$A325	TMFromScrap	I 393	\$A35E	TSGetWindowPos	I 400
\$A326	TMToScrap	I 393	\$A35F	TSCommunicate	I 400
\$A327	TMScrapHandle	I 393	\$A360	TSGetID	I 401
\$A328	TMGetScrapLen	I 393	\$A361	TSMakeEvent	I 401
\$A329	●	I 393	\$A362	●	I 401
\$A32A	TMTedTextBox	I 394	\$A363	●	I 401
\$A32B	TMTedTextBox2	I 394	\$A364	TSGetStartMode	I 402
\$A32C◎	TMCaCheON	II 287	\$A365	TSSetStartMode	I 402
\$A32D◎	TMCaCheOFF	II 287	\$A366	TMOpen	I 394
\$A32E◎	TMCaCheFlush	II 287	\$A367	TSOOpen	I 402
\$A32F◎	TMSHow	II 287	\$A368	TSClose	I 402
\$A330◎	TMHide	II 287	\$A369	TSRmDirH	I 402
\$A331◎	TMSelShow	II 287	\$A36A	TSCopyH	I 402
\$A332◎	TMSelHide	II 288	\$A36B	TSMkDirH	I 403
\$A333◎	TMSearchStrF	II 288	\$A36C	TSMoveH	I 403
\$A334◎	TMSearchStrB	II 288	\$A36D	TSCreate	I 403
\$A335◎	TMTextInWidth2	II 288	\$A36E	TSDeleteH	I 403
\$A336◎	TMTextWidth2	II 289	\$A36F	TSTrash	I 404
\$A337◎	TMDrawText2	II 289	\$A370	TSFiles	I 404
\$A338◎	TMUpdate2	II 290	\$A371	TSNFiles	I 404
\$A339◎	TMUpdate3	II 290	\$A372	TSCopyP	I 404
\$A33A◎	TMCalCOLine	II 290	\$A373	TSDeleteP	I 405
\$A33C◎	TMCalLine	II 290	\$A374	TSRmDirP	I 405
\$A33D◎	TMLeftSel	II 291	\$A375	TSMkDirP	I 405
\$A33E◎	TMRightSel	II 291	\$A376	TSMoveP	I 405
\$A33F◎	TMPointSel	II 291	\$A377	●	I 406
\$A340◎	TMOffsetSel	II 291	\$A378	TSChMod	I 406
\$A341◎	TMPointToLine	II 292	\$A379	TSWhatFile	I 406
\$A343◎	TMCalSelPoint	II 292	\$A37A	●	I 406
\$A345◎	TMSetView	II 292	\$A37B	TSDeleteVoname	I 407
\$A346◎	TMScroll	II 292	\$A37C	TSCreateVoname	I 407
\$A347◎	TMPointScroll	II 292	\$A380	●	I 407
\$A348◎	TMStr2	II 293	\$A381	TSSearchFileND	I 407
\$A349◎	TMKeyToAsk	II 293	\$A382	●	I 408
\$A34A◎	TMNextCode	II 293	\$A383	●	I 408
\$A34B◎	TMSetTextH	II 294	\$A384	●	I 408
\$A34C	TSInitTsk	I 396	\$A385	●	I 408
\$A34D	TSInitTsk2	I 396	\$A386	TSGetOpen	I 408
\$A34E	TSInitCrtM	I 396	\$A387	TSZeroDrag	I 408
\$A34F	TSInitCrtM	I 396	\$A388	TSPutDrag	I 408
\$A350	●	I 396	\$A389	TSGetDrag	I 409
\$A351	TSFock	I 397	\$A38A	TSBeginDrag	I 409
\$A352	TSExit	I 398	\$A38B	●	I 409
\$A353	TSFockB	I 398	\$A38C	TSEndDrag	I 409
\$A355	TSFockSItem	I 398	\$A38D	TSHideDrag	I 409
\$A356	TSFockIcon	I 398	\$A38E	TSShowDrag	I 409
\$A357	TSEventAvail	I 399	\$A38F	TSZeroScrap	I 409
\$A358	TSGetEvent	I 399	\$A390	TSPutScrap	I 410
\$A359	●	I 399	\$A391	TSGetScrap	I 410
\$A35A	TSPostEventTsk	I 399	\$A392	●	I 410

\$A393	●	I 410	\$A3D0	SXFilenamecmp	I 422
\$A394	●	I 410	\$A3D1	●	I 422
\$A395	●	I 410	\$A3D2	●	I 422
\$A396	●	I 411	\$A3D3	●	I 422
\$A397	TSSearchTrashpath	I 411	\$A3D4	SXSearchFname	I 423
\$A398	TSSearchTrashfile	I 411	\$A3D5	●	I 423
\$A399	TSEmptyTrash	I 411	\$A3D6	●	I 423
\$A39A	●	I 411	\$A3D7	●	I 423
\$A39B	TSSearchdpb	I 411	\$A3D8	SXStoLower	I 423
\$A39C	●	I 412	\$A3D9	SXStoUpper	I 424
\$A39D	TSDrvctrl	I 412	\$A3DA	SXStoUpper2	I 424
\$A39E	TSDrvctrl2	I 412	\$A3DB	●	I 424
\$A39F	●	I 412	\$A3DC	●	I 424
\$A3A0	●	I 413	\$A3DD	●	I 424
\$A3A1	●	I 413	\$A3DE	●	I 424
\$A3A2	SXCallWindM	I 413	\$A3DF	●	I 425
\$A3A3	SXCallCtrlM	I 413	\$A3E0	●	I 425
\$A3A4	●	I 414	\$A3E1	●	I 425
\$A3A7	●	I 414	\$A3E2	●	I 425
\$A3A8	●	I 414	\$A3E3	●	I 425
\$A3A9	●	I 414	\$A3E4	●	I 426
\$A3AA	SXInvalScBar	I 415	\$A3E5	●	I 426
\$A3AB	SXValidScBar	I 415	\$A3E6	●	I 426
\$A3AC	●	I 415	\$A3E7	●	I 426
\$A3AD	●	I 415	\$A3E8	●	I 426
\$A3AE	●	I 415	\$A3E9	SXVer	I 427
\$A3AF	●	I 415	\$A3EA	TSTakeParam	I 427
\$A3B0	●	I 416	\$A3EB	●	I 427
\$A3B1	●	I 416	\$A3EC	●	I 427
\$A3B2	●	I 417	\$A3ED	●	I 428
\$A3B4	●	I 417	\$A3EF	●	I 428
\$A3B5	●	I 417	\$A3F0	●	I 428
\$A3B7	●	I 417	\$A3F1	●	I 428
\$A3B9	●	I 418	\$A3F2	●	I 429
\$A3BB	TSISRecToStr	I 418	\$A3F3	●	I 429
\$A3BC	●	I 418	\$A3F4	TSFindTskn	I 429
\$A3BD	●	I 418	\$A3F5	●	I 429
\$A3BF	TSCreateISFile	I 418	\$A3F6	●	I 429
\$A3C0	●	I 419	\$A3F7	TSDriveCheckAll	I 429
\$A3C1	●	I 419	\$A3F8	TSDriveCheck	I 430
\$A3C2	●	I 419	\$A3F9	TSISRecToExec	I 430
\$A3C3	●	I 419	\$A3FA	TSGetDtopMode	I 430
\$A3C4	●	I 420	\$A3FB	TSSetDtopMode	I 430
\$A3C7	●	I 420	\$A3FC	TSSearchOpen	I 430
\$A3C8	●	I 420	\$A3FE	TSFindOwn	I 430
\$A3C9	●	I 420	\$A3FF	TSCommunicateS	I 431
\$A3CA	●	I 420	\$A401	TMNew2	I 395
\$A3CB	●	I 421	\$A402	TSSearchFile2	I 431
\$A3CC	SXFileConnPath	I 421	\$A403	TSSearchFile	I 432
\$A3CD	SXFileInPath	I 421	\$A404	●	I 432
\$A3CE	●	I 421	\$A405	●	I 432
\$A3CF	●	I 421	\$A406	SXStrCmp	I 432

APPENDIX

\$A407	●	I 433	\$A437◎	SXSRAMCheck	II 304
\$A408	TSCreateISBadge	I 433	ライブラリ	TSSetAbort	II 305
\$A409	●	I 433	\$A460◎	TMNextCodeIn	II 294
\$A40A	TSGetCMDs	I 433	\$A462◎	TMSelReverse	II 294
\$A40B	TSFockCM	I 434	\$A463◎	TMTini	II 294
\$A40C	●	I 434	\$A464◎	TMSetSelCal	II 294
\$A40D	TSTiniTsk	I 434	\$A465◎	TMActivate2	II 295
\$A40E	●	I 434	\$A466◎	TMDeactivate2	II 295
\$A40F	●	I 434	\$A467◎	TMCheckSel	II 295
\$A410	●	I 435	\$A468◎	TMCalPoint2	II 295
\$A411	●	I 435	\$A46A◎	TMSZen	II 296
\$A412	SXStrCopy	I 435	\$A46B◎	TMSetDestOffset	II 296
\$A413	●	I 435	\$A46C◎	TMGetDestOffset	II 296
\$A414◎	●	II 297	\$A46D◎	TMGetSelect	II 296
\$A415◎	TSPostEventTsk2	II 298	ライブラリ	TMEventW	II 297
\$A416◎	●	II 298	ライブラリ	TMUpdateExist	II 297
\$A417◎	TSAnswer	II 298	\$A4E0◎	PMInit	II 280
\$A418◎	TSSendMes	II 298	\$A4E1◎	PMTini	II 280
\$A419◎	TSGetMes	II 299	\$A4E2◎	PMOpen	II 280
\$A41A◎	TSInitTsk2	II 299	\$A4E3◎	PMClose	II 280
\$A41F◎	SXCallWindM2	II 299	\$A4E4◎	PMSetDefault	II 280
\$A420◎	TSBeginDrag2	II 300	\$A4E5◎	PMValidate	II 280
\$A421◎	●	II 300	\$A4E6◎	PMImageDialog	II 281
\$A422◎	SXGetVector	II 300	\$A4E7◎	PMStrDialog	II 281
\$A423◎	SXSetVector	II 301	\$A4E9◎	PMEnvCopy	II 281
\$A424◎	●	II 301	\$A4EA◎	PMJobCopy	II 281
\$A425◎	●	II 301	\$A4EB◎	PMOpenImage	II 281
\$A426◎	●	II 301	\$A4EC◎	PMRecordPage	II 282
\$A427◎	TSCellToStr	II 301	\$A4ED◎	PMPrintPage	II 282
\$A428◎	●	II 302	\$A4EE◎	PMCancelPage	II 282
\$A429◎	●	II 302	\$A4EF◎	PMAction	II 282
\$A42A◎	SXLockFSX	II 302	\$A4F0◎	PMCloselImage	II 282
\$A42B◎	SXUnlockFSX	II 302	\$A4F1◎	PMDrawString	II 282
\$A42C◎	TSFockMode	II 303	\$A4F2◎	PMVer	II 283
\$A42D◎	●	II 303	\$A4F3◎	PMDrvrVer	II 283
\$A42E◎	●	II 303	\$A4F4◎	PMDrvrCtrl	II 283
\$A42F◎	●	II 303	\$A4F5◎	PMDrvrID	II 283
\$A430◎	TSSetGraphMode	II 303	\$A4F6◎	PMDrvrHdl	II 283
\$A431◎	TSGetGraphMode	II 304	\$A4F7◎	PMMaxRect	II 283
\$A432◎	SXGetDispRect	II 304	\$A4F8◎	PMSaveEnv	II 284
\$A433◎	●	II 304	\$A4F9◎	PMReady	II 284
\$A434◎	●	II 304	\$A4FA◎	PMPProcPrint	II 284
\$A435◎	SXSRAMVer	II 304	\$A4FB◎	PMDrvrInfo	II 284
\$A436◎	SXSRAMReset	II 304			

コール名索引

CMCheck		\$A29A	I 379	DMWaitOpen	\$A2F7	I 386
CMDefDataGet	◎	\$A2A6	II 278	DMWaitWhile	\$A2F9	I 386
CMDefDataSet	◎	\$A2A7	II 278	EMBlinkGet	\$A0B1	I 336
CMDispose		\$A28A	I 376	EMBlinkSet	\$A0B6	I 337
CMDragControl		\$A29D	I 380	EMClean	\$A0B2	I 336
CMDraw		\$A28E	I 377	EMDClickGet	\$A0B0	I 336
CMDrawOne		\$A28F	I 377	EMDClickSet	\$A0B5	I 337
CMDraws		\$A29E	I 380	EMDeCross	\$A0B8	I 337
CMFind		\$A299	I 379	EMDTTskSet	\$A0B4	I 336
CMHide		\$A28C	I 376	EMEnCross	\$A0B7	I 337
CMKill		\$A28B	I 376	EMGet	\$A0A5	I 335
CMMaxGet		\$A295	I 378	EMInit	\$A0A2	I 334
CMMaxSet		\$A294	I 378	EMKMapGet	\$A0AE	I 336
CMMinGet		\$A293	I 378	EMLBttn	\$A0A8	I 335
CMMinSet		\$A292	I 377	EMLStill	\$A0AA	I 335
CMMove		\$A296	I 378	EMLWait	\$A0AC	I 335
CMOpen		\$A289	I 376	EMMaskSet	\$A0B3	I 336
CMOptionGet	◎	\$A2A0	II 277	EMMSLoc	\$A0A7	I 335
CMOptionSet	◎	\$A2A1	II 277	EMRBttn	\$A0A9	I 335
CMProcGet	◎	\$A2A4	II 277	EMRStill	\$A0AB	I 335
CMProcSet	◎	\$A2A5	II 277	EMRWait	\$A0AD	I 336
CMRefer		\$A29B	I 379	EMScan	\$A0A6	I 335
CMShine		\$A298	I 378	EMSet	\$A0A4	I 334
CMShow		\$A28D	I 377	EMSysTime	\$A0AF	I 336
CMSize		\$A297	I 378	EMTini	\$A0A3	I 334
CMTitleGet		\$A29C	I 380	EXAnimEnd	\$A074	I 330
CMTitleSet		\$A29F	I 380	EXAnimStart	\$A073	I 330
CMUserGet	◎	\$A2A2	II 277	EXAnimTest	\$A075	I 330
CMUserSet	◎	\$A2A3	II 277	EXDeVDISPST	\$A069	I 328
CMValueGet		\$A291	I 377	EXEnVDISPST	\$A068	I 328
CMValueSet		\$A290	I 377	GMAddFont	◎ \$A1E0	II 314
DIGet		\$A2CF	I 384	GMAdjustPt	◎ \$A1D4	II 311
DIHide		\$A2D8	I 385	GMAdjustRect	\$A159	I 348
DISet		\$A2D0	I 384	GMAAndRect	\$A154	I 347
DIShow		\$A2D9	I 386	GMAAndRectRgn	◎ \$A1D8	II 312
DITGet		\$A2D1	I 384	GMAAndRects	\$A150	I 346
DITSelect		\$A2D3	I 385	GMAAndRgn	\$A164	I 350
DITSet		\$A2D2	I 385	GMAPage	\$A149	I 345
DIUpdate		\$A2D6	I 385	GMBBackColor	\$A148	I 344
DMBeep		\$A2D7	I 385	GMCalcBitmap	◎ \$A1C8	II 309
DMClose		\$A2C5	I 382	GMCalcFrame	\$A1B2	I 364
DMControl		\$A2C7	I 382	GMCalcGraph	◎ \$A1D1	II 311
DMDDispose		\$A2C6	I 382	GMCalcMask	\$A1B1	I 363
DMDraw		\$A2C8	I 382	GMCalcScrnSize	◎ \$A1C9	II 309
DMEError		\$A2F6	I 386	GMCenterRect	◎ \$A1D6	II 312
DMFontSet		\$A2C2	I 381	GMCharKind	◎ \$A1DC	II 313
DMInit		\$A2C0	I 380	GMCharWidth	\$A194	I 358
DMOpen		\$A2C3	I 381	GMClipRect	\$A13A	I 342
DMRefer		\$A2C4	I 382	GMCloseGraph	\$A12E	I 340
DMWaitClose		\$A2F8	I 386	GMClosePoly	◎ \$A19F	II 306

APPENDIX

GMCloseRgn		\$A15D	I 348	GMFreeBits	⊙	\$A1D0	II311
GMCloseScript		\$A19A	I 359	GMGetBitmap	⊙	\$A1C7	II309
GMCopy		\$A17F	I 354	GMGetClip		\$A139	I 342
GMCopyGraph		\$A133	I 341	GMGetFontLink	⊙	\$A1E2	II314
GMCopyMask		\$A180	I 354	GMGetFontProcTbl	⊙	\$A1E7	II315
GMCopyRgn		\$A160	I 349	GMGetFontTable		\$A1B8	I 365
GMCopyStdProc	⊙	\$A1B9	II307	GMGetGraph		\$A132	I 341
GMCurFont	⊙	\$A1C3	II309	GMGetHProcTbl	⊙	\$A1E3	II314
GMDiffRectRgn	⊙	\$A1DA	II313	GMGetLoc		\$A14A	I 345
GMDiffRgn		\$A166	I 350	GMGetPen		\$A14B	I 345
GMDiffRgnRect	⊙	\$A1DD	II314	GMGetPixel		\$A1AF	I 363
GMDisposeBits	⊙	\$A1CB	II310	GMGetRgnLine	⊙	\$A1C1	II308
GMDisposePoly	⊙	\$A1A0	II306	GMGetRgnProcTbl	⊙	\$A1E8	II315
GMDisposeRgn		\$A15B	I 348	GMGetScript		\$A19D	I 360
GMDisposeScript		\$A19B	I 360	GMGetScrnSize	⊙	\$A1C4	II309
GMDrawChar		\$A18F	I 357	GMGetStdProcTbl	⊙	\$A1E6	II315
GMDrawG16		\$A1AD	I 363	GMGlobalToLocal		\$A13F	I 343
GMDrawScript		\$A19C	I 360	GMImgToRgn	⊙	\$A16C	II305
GMDrawStr		\$A191	I 358	GMInitBitmap		\$A16D	I 352
GMDrawStrL		\$A190	I 358	GMInitGraph		\$A130	I 341
GMDrawStrZ		\$A192	I 358	GMInitGraphMode	⊙	\$A1C2	II308
GMDupHImg		\$A186	I 356	GMInitialize		\$A14D	I 345
GMDupHRImg		\$A188	I 356	GMInitPalet		\$A1AB	I 362
GMDupVImg		\$A187	I 356	GMInitPen		\$A140	I 343
GMDupVRImg		\$A189	I 356	GMInsetRect		\$A153	I 346
GMEmpyRect		\$A158	I 348	GMInsetRgn		\$A163	I 349
GMEmpyRgn		\$A16B	I 351	GMInvertBits		\$A1A5	I 361
GMEqualRect		\$A157	I 347	GMInvertRect		\$A1A3	I 361
GMEqualRgn		\$A16A	I 351	GMItalicRect	⊙	\$A1CE	II310
GMExgBitmap	⊙	\$A1C6	II309	GMItalicRgn	⊙	\$A1CF	II310
GMExgGraph	⊙	\$A1C5	II309	GMLine		\$A170	I 352
GMExPat		\$A146	I 344	GMLineRel		\$A171	I 352
GMFillArc	⊙	\$A179	II305	GMLocalToGlobal		\$A13E	I 343
GMFillImg	⊙	\$A1BD	II307	GMLockBits	⊙	\$A1CC	II310
GMFillOval		\$A175	I 353	GMMMapPoly	⊙	\$A1A8	II306
GMFillPoly	⊙	\$A17D	II306	GMMMapPt		\$A1A6	I 361
GMFillRect		\$A173	I 353	GMMMapRect		\$A1A7	I 361
GMFillRgn		\$A17B	I 354	GMMMapRgn		\$A1A9	I 362
GMFillRImg	⊙	\$A1BC	II307	GMMove		\$A16E	I 352
GMFillRRect		\$A177	I 353	GMMoveGraph		\$A136	I 341
GMFontFace		\$A18C	I 357	GMMoveRect		\$A151	I 346
GMFontInfo		\$A198	I 359	GMMoveRel		\$A16F	I 352
GMFontKind		\$A18B	I 357	GMMoveRgn		\$A161	I 349
GMFontMode		\$A18D	I 357	GMNewBits	⊙	\$A1CA	II310
GMFontSize		\$A18E	I 357	GMNewRgn		\$A15A	I 348
GMForeColor		\$A147	I 344	GMNullRect		\$A14E	I 345
GMFrameArc	⊙	\$A178	II305	GMNullRgn		\$A15E	I 349
GMFrameOval		\$A174	I 353	GMOpenGraph		\$A12D	I 340
GMFramePoly	⊙	\$A17C	II306	GMOpenPoly	⊙	\$A19E	II306
GMFrameRect		\$A172	I 352	GMOpenRgn		\$A15C	I 348
GMFrameRgn		\$A17A	I 353	GMOpenScript		\$A199	I 359
GMFrameRRect		\$A176	I 353	GMOOrRect		\$A155	I 347

GMOrectRgn	◎	\$AID9	II313	KBScan	\$A08A	I 332
GMOrectRgn		\$A165	I 350	KBShiftGet	\$A087	I 331
GMPackImage	◎	\$AID2	II311	KBShiftSet	\$A088	I 331
GMPaintRgn	◎	\$A1BF	II308	KBSimulate	\$A089	I 332
GMPenHide		\$A142	I 343	KBTini	\$A08E	I 331
GMPenMode		\$A144	I 344	KMAscJobSet	\$A09C	I 333
GMPenPat		\$A145	I 344	KMCurKmrGet	\$A0A1	I 334
GMPenShow		\$A141	I 343	KMEmpty	\$A09A	I 333
GMPenSize		\$A143	I 343	KMInit	\$A09F	I 333
GMPlotImg		\$A182	I 355	KMPost	\$A09B	I 333
GMPtlnRect		\$A156	I 347	KMSimulate	\$A09D	I 333
GMPtlnRgn		\$A168	I 351	KMTask	\$A09E	I 333
GMPutImg	◎	\$AID5	II312	KMTini	\$A0A0	I 334
GMPutRimg		\$A183	I 355	MMBlockMstGet	\$A00F	I 317
GMRectlnRgn		\$A169	I 351	MMBlockUsrFlagGet	\$A048	I 326
GMRectRgn		\$A15F	I 349	MMBlockUsrFlagSet	\$A049	I 326
GMRemoveFont	◎	\$A1E1	II314	MMBlockUsrWordGet	\$A04A	I 326
GMScalePt		\$A1AA	I 362	MMBlockUsrWordSet	\$A04B	I 326
GMscrewRect	◎	\$AID7	II312	MMChCompact	\$A022	I 320
GMScroll		\$A17E	I 354	MMChCompactGet	\$A02B	I 321
GMSetBitmap		\$A13D	I 343	MMChCompactSet	\$A02C	I 321
GMSetClip		\$A138	I 342	MMChFreeSize	\$A026	I 321
GMSetGraph		\$A131	I 341	MMChGet	\$A01C	I 319
GMSetGraphSize		\$A13C	I 342	MMChGrowHeapGet	\$A027	I 321
GMSetHome		\$A13B	I 342	MMChGrowHeapSet	\$A028	I 321
GMSetPen		\$A14C	I 345	MMChHdlNew	\$A021	I 320
GMSetRgnLine	◎	\$A1C0	II308	MMChHiReserve	\$A051	I 328
GMShadowRect		\$A1A2	I 361	MMChMelt	\$A024	I 320
GMShadowStrZ		\$A1A1	I 360	MMChMstMore	\$A01F	I 320
GMSizeRect		\$A14F	I 345	MMChMstNew	\$A020	I 320
GMSlidedRgn	◎	\$A1BE	II308	MMChPtrNew	\$A01E	I 319
GMSlideGraph		\$A137	I 342	MMChPurge	\$A023	I 320
GMSlideRect		\$A152	I 346	MMChPurgeGet	\$A029	I 321
GMSlideRgn		\$A162	I 349	MMChPurgeSet	\$A02A	I 321
GMStrLength		\$A197	I 359	MMChReserve	\$A025	I 321
GMStrLWidth		\$A195	I 358	MMChSet	\$A01D	I 319
GMStrWidth		\$A196	I 359	MMChUsrFlagGet	\$A052	I 328
GMStrZWidth		\$A1BA	I 365	MMChUsrFlagSet	\$A053	I 328
GMTranslmg	◎	\$A1BB	II307	MMChUsrWordGet	\$A054	I 328
GMUnlockBits	◎	\$A1CD	II310	MMChUsrWordSet	\$A055	I 328
GMUnpackImage	◎	\$AID3	II311	MMCompactMem	\$A00D	I 317
GMXorRectRgn	◎	\$AIDB	II313	MMDisposeHandle	\$A005	I 315
GMXorRgn		\$A167	I 350	MMDisposePtr	\$A00A	I 316
KBCurKbrGet		\$A08F	I 332	MMGetCurrentHeap	\$A001	I 315
KBEmpty		\$A08C	I 331	MMGetHandleSize	\$A006	I 316
KBFlagGet	◎	\$A092	II275	MMGetPtrSize	\$A00B	I 316
KBFlagSet	◎	\$A093	II275	MMHdlBlock	\$A04F	I 327
KBGet		\$A08B	I 332	MMHdlDel	\$A047	I 326
KBInit		\$A08D	I 331	MMHdlDispose	\$A038	I 323
KBMapGet		\$A086	I 331	MMHdlEmpty	\$A03B	I 324
KBOldOnGet		\$A090	I 332	MMHdlHeap	\$A037	I 323
KBOldOnSet		\$A091	I 332	MMHdlIns	\$A046	I 326

APPENDIX

MMHdlLock	\$A040	I 325	MSHideCsr	\$A06C	I 329
MMHdlMoveHi	\$A03D	I 324	MSInitCsr	\$A06A	I 329
MMHdlMstGet	\$A050	I 327	MSMultiGet	\$A071	I 330
MMHdlNew	\$A036	I 323	MSMultiSet	\$A072	I 330
MMHdlNoPurge	\$A043	I 325	MSObscureCsr	\$A06E	I 329
MMHdlNoResource	\$A045	I 325	MSSetCsr	\$A06D	I 329
MMHdlPropGet	\$A03E	I 324	MSShieldCsr	\$A06F	I 329
MMHdlPropSet	\$A03F	I 324	MSShowCsr	\$A06B	I 329
MMHdlPurge	\$A042	I 325	PMAction	⊙ \$A4EF	II 282
MMHdlRealloc	\$A03C	I 324	PMCancelPage	⊙ \$A4EE	II 282
MMHdlResource	\$A044	I 325	PMCclose	⊙ \$A4E3	II 280
MMHdlSizeGet	\$A039	I 323	PMCcloseImage	⊙ \$A4F0	II 282
MMHdlSizeSet	\$A03A	I 324	PMDrawString	⊙ \$A4F1	II 282
MMHdlUnlock	\$A041	I 325	PMDrvrCtrl	⊙ \$A4F4	II 283
MMHeapInit	\$A00E	I 317	PMDrvrHdl	⊙ \$A4F6	II 283
MMHLock	\$A007	I 316	PMDrvrID	⊙ \$A4F5	II 283
MMHUnlock	\$A008	I 316	PMDrvrInfo	⊙ \$A4FB	II 284
MMInitHeap	\$A000	I 315	PMDrvrVer	⊙ \$A4F3	II 283
MMMemAmiTPeach	\$A04C	I 327	PMEnvCopy	⊙ \$A4E9	II 281
MMMemCompact	\$A010	I 317	PMImageDialog	⊙ \$A4E6	II 281
MMMemErrorGet	\$A018	I 319	PMInit	⊙ \$A4E0	II 280
MMMemErrorSet	\$A019	I 319	PMJobCopy	⊙ \$A4EA	II 281
MMMemHiReserve	\$A04D	I 327	PMMaxRect	⊙ \$A4F7	II 283
MMMemMelt	\$A012	I 318	PMOpen	⊙ \$A4E2	II 280
MMMemPurge	\$A011	I 317	PMOpenImage	⊙ \$A4EB	II 281
MMMemReserve	\$A013	I 318	PMPrintPage	⊙ \$A4ED	II 282
MMMemSizeComp	\$A015	I 318	PMProcPrint	⊙ \$A4FA	II 284
MMMemSizeFree	\$A014	I 318	PMReady	⊙ \$A4F9	II 284
MMMemSizeMelt	\$A017	I 318	PMRecordPage	⊙ \$A4EC	II 282
MMMemSizePurg	\$A016	I 318	PMSaveEnv	⊙ \$A4F8	II 284
MMMemStrictGet	\$A01A	I 319	PMSetDefault	⊙ \$A4E4	II 280
MMMemStrictSet	\$A01B	I 319	PMStrDialog	⊙ \$A4E7	II 281
MMMstAllocate	\$A034	I 323	PMTini	⊙ \$A4E1	II 280
MMMstBind	\$A035	I 323	PMValidate	⊙ \$A4E5	II 280
MMNewHandle	\$A003	I 315	PMVer	⊙ \$A4F2	II 283
MMNewPtr	\$A009	I 316	RMCurResGet	\$A0EA	I 340
MMPtrBlock	\$A04E	I 327	RMCurResSet	\$A0E4	I 339
MMPtrDispose	\$A02F	I 322	RMHdlToRsc	\$A0E9	I 339
MMPtrHeap	\$A02E	I 322	RMInit	\$A0D9	I 337
MMPtrNew	\$A02D	I 321	RMLastResGet	\$A0EB	I 340
MMPtrPropGet	\$A032	I 322	RMMaxIDGet	\$A0E7	I 339
MMPtrPropSet	\$A033	I 322	RMResClose	\$A0E2	I 338
MMPtrSizeGet	\$A030	I 322	RMResDispose	\$A0DF	I 338
MMPtrSizeSet	\$A031	I 322	RMResIDList	⊙ \$A0EF	II 276
MMSetCurrentHeap	\$A002	I 315	RMResLinkGet	⊙ \$A0ED	II 275
MMSetHandleSize	\$A004	I 315	RMResLoad	\$A0EC	I 340
MMSetPtrSize	\$A00C	I 316	RMResNew	\$A0DB	I 337
MNConvert	⊙ \$A269	II 278	RMResOpen	\$A0E0	I 338
MNInit	\$A266	I 375	RMResRemove	\$A0E3	I 339
MNRefer	\$A267	I 375	RMResSave	\$A0E8	I 339
MNSelect	\$A268	I 375	RMResTypeList	⊙ \$A0EE	II 275
MSGetCurMsr	\$A070	I 330	RMRscAdd	\$A0DC	I 337

RMRscDetach	\$A0E6	I 339	TMDispose	\$A312	I 388
RMRscGet	\$A0E1	I 338	TMDrawText2	◎ \$A337	II289
RMRscRelease	\$A0E5	I 339	TMEvent	◎ \$A31C	(I 391), II285
RMRscRemove	\$A0DD	I 338	TMEventW	ライブラリ	II297
RMTini	\$A0DA	I 337	TMFromScrap	\$A325	I 393
RMTTypeRemove	\$A0DE	I 338	TMGetDestOffset	◎ \$A46C	II296
SXCallCtrlM	\$A3A3	I 413	TMGetScrapLen	\$A328	I 393
SXCallWindM	\$A3A2	I 413	TMGetSelect	◎ \$A46D	II296
SXCallWindM2	◎ \$A41F	II299	TMGetText	\$A315	I 389
SXFileConnPath	\$A3CC	I 421	TMHide	◎ \$A330	II287
SXFileInPath	\$A3CD	I 421	TMIdle	\$A30E	I 388
SXFixDiv	\$A1B7	I 365	TMInit	\$A30A	I 387
SXFixMul	\$A1B6	I 364	TMInsert	◎ \$A324	(I 393), II286
SXFixRound	\$A1B4	I 364	TMISSzen	◎ \$A46A	II296
SXFilenamecmp	\$A3D0	I 422	TMKey	◎ \$A317	(I 390), II285
SXGetDispRect	◎ \$A432	II304	TMKeyToAsk	◎ \$A349	II293
SXGetVector	◎ \$A422	II300	TMLeftSel	◎ \$A33D	II291
SXInvalScBar	\$A3AA	I 415	TMNew	\$A30B	I 387
SXLockFSX	◎ \$A42A	II302	TMNew2	\$A401	I 395
SXLongMul	\$A1B3	I 364	TMNextCode	◎ \$A34A	II293
SXSearchFname	\$A3D4	I 423	TMNextCodeIn	◎ \$A460	II294
SXSetVector	◎ \$A423	II301	TMOffsetSel	◎ \$A340	II291
SXSRAMCheck	◎ \$A437	II304	TMOpen	\$A366	I 394
SXSRAMReset	◎ \$A436	II304	TMPaste	◎ \$A322	(I 392), II286
SXSRAMVer	◎ \$A435	II304	TMPinScroll	\$A31A	I 390
SXStoLower	\$A3D8	I 423	TMPointScroll	◎ \$A347	II292
SXStoUpper	\$A3D9	I 424	TMPointSel	◎ \$A33F	II291
SXStoUpper2	\$A3DA	I 424	TMPointToLine	◎ \$A341	II292
SXStrCmp	\$A406	I 432	TMRightSel	◎ \$A33E	II291
SXStrCopy	\$A412	I 435	TMScrapHandle	\$A327	I 393
SXUnlockFSX	◎ \$A42B	II302	TMScroll	◎ \$A346	II292
SXValidScBar	\$A3AB	I 415	TMSearchStrB	◎ \$A334	II288
SXVer	\$A3E9	I 427	TMSearchStrF	◎ \$A333	II288
TMActivate2	◎ \$A465	II295	TMSelHide	◎ \$A332	II288
TMActive	\$A30F	I 388	TMSelReverse	◎ \$A462	II294
TMCacheFlush	◎ \$A32E	II287	TMSelShow	◎ \$A331	II287
TMCacheOFF	◎ \$A32D	II287	TMSetDestOffset	◎ \$A46B	II296
TMCacheON	◎ \$A32C	II287	TMSetRect	\$A30C	I 387
TMCalCOLine	◎ \$A33A	II290	TMSetSelCal	◎ \$A464	II294
TMCalLine	◎ \$A33C	II290	TMSetSelect	\$A316	I 389
TMCalPoint2	◎ \$A468	II295	TMSetText	\$A314	I 389
TMCalSelPoint	◎ \$A343	II292	TMSetTextH	◎ \$A34B	II294
TMCalText	◎ \$A319	(I 390), II285	TMSetView	◎ \$A345	II292
TM caret	\$A311	I 388	TMShow	◎ \$A32F	II287
TMCheckSel	◎ \$A467	II295	TMStr	◎ \$A318	(I 390), II285
TMChangText	\$A30D	I 388	TMStr2	◎ \$A348	II293
TMClick	\$A31B	I 390	TMTTextBox	\$A32A	I 394
TMCopy	\$A321	I 392	TMTTextBox2	\$A32B	I 394
TMCut	◎ \$A320	(I 392), II286	TMTTextInWidth2	◎ \$A335	II288
TMDeactivate2	◎ \$A466	II295	TMTTextWidth2	◎ \$A336	II289
TMDeactive	\$A310	I 388	TMTini	◎ \$A463	II294
TMDDelete	◎ \$A323	(I 393), II286	TMTToScrap	\$A326	I 393

APPENDIX

TMUpDate		\$A313	I 389	TSInitTsk		\$A34C	I 396
TMUpDate2	◎	\$A338	II 290	TSInitTsk2	◎	\$A41A	II 299
TMUpDate3	◎	\$A339	II 290	TSISRecToExec		\$A3F9	I 430
TMUpDateExist		ライブラリ	II 297	TSISRecToStr		\$A3BB	I 418
TSAnswer	◎	\$A417	II 298	TSMakeEvent		\$A361	I 401
TSEginDrag		\$A38A	I 409	TSMkDirH		\$A36B	I 403
TSEginDrag2	◎	\$A420	II 300	TSMkDirP		\$A375	I 405
TSCellToStr	◎	\$A427	II 301	TSMoveH		\$A36C	I 403
TSchMod		\$A378	I 406	TSMoveP		\$A376	I 405
TSClose		\$A368	I 402	TSNFiles		\$A371	I 404
TSCommunicate		\$A35F	I 400	TSEopen		\$A367	I 402
TSCommunicateS		\$A3FF	I 431	TSPostEventTsk		\$A35A	I 399
TSCopyH		\$A36A	I 402	TSPostEventTsk2	◎	\$A415	II 298
TSCopyP		\$A372	I 404	TSPutDrag		\$A388	I 408
TSCreate		\$A36D	I 403	TSPutScrap		\$A390	I 410
TSCreateISBadge		\$A408	I 433	TSRmDirH		\$A369	I 402
TSCreateISFile		\$A3BF	I 418	TSRmDirP		\$A374	I 405
TSCreateVoname		\$A37C	I 407	TSSearchdpb		\$A39B	I 411
TSDeleteH		\$A36E	I 403	TSSearchFile		\$A403	I 432
TSDeleteP		\$A373	I 405	TSSearchFile2		\$A402	I 431
TSDeleteVoname		\$A37B	I 407	TSSearchFileND		\$A381	I 407
TSDriveCheck		\$A3F8	I 430	TSSearchOpen		\$A3FC	I 430
TSDriveCheckAll		\$A3F7	I 429	TSSearchTrashfile		\$A398	I 411
TSDrvctrl		\$A39D	I 412	TSSearchTrashpath		\$A397	I 411
TSDrvctrl2		\$A39E	I 412	TSSendMes	◎	\$A418	II 298
TSEmptyTrash		\$A399	I 411	TSSetAbort		ライブラリ	II 305
TSEndDrag		\$A38C	I 409	TSSetDtopMode		\$A3FB	I 430
TSEventAvail		\$A357	I 399	TSSetGraphMode	◎	\$A430	II 303
TSExit		\$A352	I 398	TSSetStartMode		\$A365	I 402
TSFiles		\$A370	I 404	TSSetTdb		\$A35C	I 400
TSFindOwn		\$A3FE	I 430	TSShowDrag		\$A38E	I 409
TSFindTskn		\$A3F4	I 429	TSTakeParam		\$A3EA	I 427
TSFock		\$A351	I 397	TSIniCrtM		\$A34F	I 396
TSFockB		\$A353	I 398	TSIniTsk		\$A40D	I 434
TSFockCM		\$A40B	I 434	TSIniTsk2		\$A34D	I 396
TSFockIcon		\$A356	I 398	TSTrash		\$A36F	I 404
TSFockMode	◎	\$A42C	II 303	TSWhatFile		\$A379	I 406
TSFockSItem		\$A355	I 398	TSZeroDrag		\$A387	I 408
TSGetCMDS		\$A40A	I 433	TSZeroScrap		\$A38F	I 409
TSGetDrag		\$A389	I 409	WMActive		\$A20F	I 371
TSGetDtopMode		\$A3FA	I 430	WMAddRect		\$A218	I 372
TSGetEvent		\$A358	I 399	WMAddRgn		\$A219	I 373
TSGetGraphMode	◎	\$A431	II 304	WMCalcUpdt		\$A223	I 374
TSGetID		\$A360	I 401	WMCarry		\$A200	I 368
TSGetMes	◎	\$A419	II 299	WMCheckBox		\$A20A	I 370
TSGetOpen		\$A386	I 408	WMCheckCBox		\$A20B	I 370
TSGetScrap		\$A391	I 410	WMClose		\$A1FB	I 367
TSGetStartMode		\$A364	I 402	WMDispose		\$A1FC	I 367
TSGetTdb		\$A35B	I 400	WMDrag		\$A205	I 369
TSGetWindowPos		\$A35E	I 400	WMDragRgn		\$A225	I 374
TSHideDrag		\$A38D	I 409	WMDrawGBox		\$A20C	I 370
TSInitCrtM		\$A34E	I 396	WMFind		\$A1FD	I 367

WMGetDTGS		\$A224	I 374	●	\$A2D5	I 385
WMGraphGet		\$A210	I 371	●	\$A31D	I 391
WMGrow		\$A204	I 369	●	\$A31E	I 392
WMGScriptGet		\$A21D	I 373	●	\$A31F	I 392
WMGScriptSet		\$A21C	I 373	●	\$A329	I 393
WMHide		\$A208	I 370	●	\$A350	I 396
WMInit		\$A1F8	I 366	●	\$A359	I 399
WMMove		\$A202	I 368	●	\$A362	I 401
WMOpen		\$A1F9	I 366	●	\$A363	I 401
WMOptionGet	◎	\$A22C	II276	●	\$A377	I 406
WMOptionSet	◎	\$A22D	II276	●	\$A37A	I 406
WMPinRect		\$A222	I 374	●	\$A380	I 407
WMRefer		\$A1FA	I 366	●	\$A382	I 408
WMSelect		\$A1FE	I 367	●	\$A383	I 408
WMSelect2	◎	\$A1FF	(I 367), II276	●	\$A384	I 408
WMShine		\$A201	I 368	●	\$A385	I 408
WMShow		\$A207	I 369	●	\$A38B	I 409
WMShowHide		\$A209	I 370	●	\$A392	I 410
WMSize		\$A203	I 368	●	\$A393	I 410
WMSubRect		\$A21A	I 373	●	\$A394	I 410
WMSubRgn		\$A21B	I 373	●	\$A395	I 410
WMTIDGet		\$A221	I 374	●	\$A396	I 411
WMTIDSet		\$A220	I 374	●	\$A39A	I 411
WMTitleGet		\$A21F	I 373	●	\$A39C	I 412
WMTitleSet		\$A21E	I 373	●	\$A39F	I 412
WMUpdate		\$A20D	I 371	●	\$A3A0	I 413
WMUpdtOver		\$A20E	I 371	●	\$A3A1	I 413
WMZoom		\$A206	I 369	●	\$A3A4	I 414
WSClose	◎	\$A228	II279	●	\$A3A7	I 414
WSDelist	◎	\$A22B	II279	●	\$A3A8	I 414
WSDispose	◎	\$A229	II279	●	\$A3A9	I 414
WSEnlist	◎	\$A22A	II279	●	\$A3AC	I 415
WSOpen	◎	\$A227	II279	●	\$A3AD	I 415
●		\$A135	I 341	●	\$A3AE	I 415
●		\$A181	I 355	●	\$A3AF	I 415
●		\$A1AC	I 363	●	\$A3B0	I 416
●		\$A1AE	I 363	●	\$A3B1	I 416
●		\$A211	I 371	●	\$A3B2	I 417
●		\$A212	I 371	●	\$A3B4	I 417
●		\$A213	I 371	●	\$A3B5	I 417
●		\$A214	I 372	●	\$A3B7	I 417
●		\$A215	I 372	●	\$A3B9	I 418
●		\$A216	I 372	●	\$A3BC	I 418
●		\$A217	I 372	●	\$A3BD	I 418
●		\$A2C1	I 381	●	\$A3C0	I 419
●		\$A2C9	I 383	●	\$A3C1	I 419
●		\$A2CA	I 383	●	\$A3C2	I 419
●		\$A2CB	I 383	●	\$A3C3	I 419
●		\$A2CC	I 383	●	\$A3C4	I 420
●		\$A2CD	I 383	●	\$A3C7	I 420
●		\$A2CE	I 384	●	\$A3C8	I 420
●		\$A2D4	I 385	●	\$A3C9	I 420

APPENDIX

●	\$A3CA	I 420	●	\$A3F0	I 428
●	\$A3CB	I 421	●	\$A3F1	I 428
●	\$A3CE	I 421	●	\$A3F2	I 429
●	\$A3CF	I 421	●	\$A3F3	I 429
●	\$A3D1	I 422	●	\$A3F5	I 429
●	\$A3D2	I 422	●	\$A3F6	I 429
●	\$A3D3	I 422	●	\$A404	I 432
●	\$A3D5	I 423	●	\$A405	I 432
●	\$A3D6	I 423	●	\$A407	I 433
●	\$A3D7	I 423	●	\$A409	I 433
●	\$A3DB	I 424	●	\$A40C	I 434
●	\$A3DC	I 424	●	\$A40E	I 434
●	\$A3DD	I 424	●	\$A40F	I 434
●	\$A3DE	I 424	●	\$A410	I 435
●	\$A3DF	I 425	●	\$A411	I 435
●	\$A3E0	I 425	●	\$A413	I 435
●	\$A3E1	I 425	●	◎ \$A414	II 297
●	\$A3E2	I 425	●	◎ \$A416	II 298
●	\$A3E3	I 425	●	◎ \$A421	II 300
●	\$A3E4	I 426	●	◎ \$A424	II 301
●	\$A3E5	I 426	●	◎ \$A425	II 301
●	\$A3E6	I 426	●	◎ \$A426	II 301
●	\$A3E7	I 426	●	◎ \$A428	II 302
●	\$A3E8	I 426	●	◎ \$A429	II 302
●	\$A3EB	I 427	●	◎ \$A42D	II 303
●	\$A3EC	I 427	●	◎ \$A42E	II 303
●	\$A3ED	I 428	●	◎ \$A42F	II 303
●	\$A3EF	I 428	●	◎ \$A433	II 304
			●	◎ \$A434	II 304



あとがき

前著『SX-WINDOW プログラミング』刊行から半年以上経ったいま、決して心底満足するほどではないにしろ、SX-WINDOW をとりまく状況が確実に変化してきたことを感じています。おそらくは、よい方向に。

来春にはシャープより純正 SX-WINDOW 開発キットがリリースされる予定ですし、サードパーティからも SX 対応ソフトの発売が予定されている模様です。市販ソフトの露払い役を務めた EasyPaint も素敵なソフトでした。

パワーユーザの存在なくしては語れない X68000 の世界では、彼らが力を貸してくれるか否かで物事の動きはずいぶん違ってきます。パソコン通信等で SX-WINDOW のフリーソフトウェアを公開してくれるパワーユーザ（誰が呼んだか「SXer」）の皆さんの存在は心強いばかりです。そのパワーの一部は、付録ディスクに収録させていただいたフリーソフトウェアから感じ取っていただけたと思います。

SX1.10 が、こうした動きを支えるだけの地力を備えていることは、本書でいくらかはお伝えできたのではないかと思います。また、SX1.02 から SX1.10 へのバージョンアップがそうであったように、新たなニーズに対応するために、遠からず、さらなる機能強化が行われることも考えられます。

新しいプラットフォームとしての SX-WINDOW は、いよいよ本格的な離陸の時期を迎えたようです。より高度なアプリケーションが作成できることはもちろん、より容易に、誰にでもアプリケーションが作成できる環境が整うことも期待したいところです。

本書では SX-WINDOW の「現在」をお伝えしました。

本書をもって SX-WINDOW の「未来」を創る皆さんのお手伝いができるのなら、私にとって無上の喜びです。

最後に、前著および本書の執筆にあたり、ご協力いただいたすべての方に心から感謝いたします。

1991 年 11 月

吉沢正敏

I N D E X ● 欧文

A

arc▶109

D

DB.X▶90, 99, 104

E

Easypaint▶14, 150, 206

G

GCC▶269

GUI▶20

S

SRAM▶142, 143

static▶249, 250, 251

SXDEF.H▶247

SXKERNEL.X▶39, 89, 91, 99

SXLIB.H▶247, 256

SXLIB.L▶247, 249, 251, 253, 256

SXWDB.X▶91, 99, 103

SXWIN.X▶15

SYSDDTOP.SX▶61, 64, 144

X

XC2▶247, 249, 253

I N D E X ● 和文

あ 行

イベント▶20, 23, 47, 53, 257

印刷環境設定ダイアログ▶173, 189

印刷環境レコード▶175, 186, 189, 191

エディタ.X▶13

円弧▶109, 151

か 行

ガイドライン▶22, 29, 38

外部カーネル▶89, 91, 99

画面モード▶111, 142

疑似ダイアログ▶114

キャッシュ▶128

行▶123

行頭テーブル▶127

グラフポート▶151, 173, 183, 184, 185, 191, 194

グローバル変数▶249, 250, 251

高級アセンブラ▶246

コード印刷▶171, 188

コモンエリア▶138, 139, 250

コントロールキー処理ルーチンテーブル▶126

コントロールキーテーブル▶126

さ 行

サブウィンドウ▶206, 207, 215, 230

サブウィンドウリスト▶210, 212, 213, 232

サブウィンドウレコード▶208

実画面モード▶142

主ウィンドウ▶207, 212, 213

スクリーンタイプ▶106

スケルトン▶38, 255, 257, 262

スタックチェックオプション▶251

スタートアップ▶247, 249, 251, 253, 254, 256

セル▶140, 152

⑨ 行

タスク間通信▶139
 タスク管理テーブル▶62, 64, 139
 タスクマネージャイベント▶140
 ターミナル▶91, 102
 段落▶123
 段落情報▶127
 テキストエディット▶123, 262
 デバッグ▶90, 92, 95, 103
 デバッグ用カーネル▶91
 デバッグ用スケルトン▶99
 ドローレベル▶125

⑨ 行

ハイライト表示レベル▶125
 ビッツ▶106, 151
 ビットマップ▶106, 108, 151, 174, 184, 185, 192, 193, 194,
 222, 223
 ヒープ▶249, 250, 251
 ファンクションキーアサインテーブル▶126
 フォント▶111
 フォントリンク▶111
 フォントレコード▶114

ブライオリティ▶209, 210, 212
 プリントドライバ▶171, 180, 188, 191, 198
 プリントマネージャ▶14, 170
 プロセス印刷▶174, 193, 221
 プロセステーブル▶125, 130
 ペクタ▶142
 ページ印刷▶173, 191
 ヘッドファイル▶247
 編集モード▶125
 編集履歴▶128
 ポリゴン▶109, 110

⑨ 行

モジュールタイプ▶137
 モジュールヘッド▶137

⑨ 行

ユーザーインタフェース▶22

⑨ 行

ライブラリ▶247, 253, 254, 271
 リソース▶135, 141, 170, 174, 180, 189, 195, 197
 リファレンス▶256

参考文献

- 1) 『SX-WINDOW ユーザーズマニュアル』, SHARP
- 2) 『プログラマーズマニュアル』, SHARP
- 3) 『Easypaint ユーザーズマニュアル』, SHARP
- 4) 『68000 プログラマーズハンドブック』, 実倉幸則, 技術評論社
- 5) 『Oh!X』 '90 年 1 月号/付録ディスク, ソフトバンク
- 6) BNN 第 2 企画部, 『インサイドマック徹底ガイド 上巻/下巻』, BNN
- 7) 『別冊インタフェース・Macintosh 活用ハンドブック』, CQ 出版
- 8) 市原昌文/吉沢正敏, 『X68000 環境ハンドブック』, 工学社

追補版 SX-WINDOW プログラミング

1991 年 12 月 20 日 初版第 1 刷印刷

1991 年 12 月 25 日 初版第 1 刷発行

編 者 吉沢正敏

発行者 孫 正義

発行所 ソフトバンク株式会社 出版事業部

〒108 東京都港区高輪 2-19-13 NS 高輪ビル

営業部 03 (5488) 1360

編集部 03 (5488) 1326

印刷所 株式会社厚德社

編 集 (有) ガジェット

© M. YOSHIZAWA 1991 Printed in JAPAN.

ISBN4-89052-284-0 C0055

落丁、乱丁はお取り替え致します。

定価はカバーに表示してあります。







ソフトバンク

定価4200円(本体4078円 フロッピーディスク含む)

ISBN4-89052-284-0 C0055 P4200E

前著『SX-WINDOWプログラミング』刊行後、発売されたSX-WINDOW ver. 1.10は、画面描画スピードの向上、プリンタマネージャ/プリンタドライバ周辺の充実、そして優秀なエディタの添付など、さらに実用性が高められた内容となっている。本書は、この新しいSX-WINDOW ver. 1.10に対応すべく書き下ろされたものである。記述のポイントは、大幅に増設されたSXコール、新設された2つのマネージャの解説のほか、C言語でのプログラミングについても触れている。また、付録ディスクには、前著と本書で取り上げたサンプルプログラムのほか、ver. 1.10対応のCのライブラリ(サンプル版)、PDSとして流布されているSX-WINDOW上のアプリケーションも収録している。

本書の内容

第0章 SX-WINDOW ver. 1.10の概要

第1章 プログラミングの補足説明

第2章 拡張されたマネージャ

第3章 新設されたマネージャ

第4章 C言語によるプログラミング

第5章 SXコールリファレンス

APPENDIX 付録ディスク「SXer Tool Box」の使い方

SX1.10/Easy Paintで追加されたリソース

リザルトコード一覧

SXコール通巻索引